

4/5/1 (Item 1 from file: 351)
DIALOG(R) File 351: Derwent WPI
(c) 2004 Thomson Derwent. All rts. reserv.

014185236 **Image available**
WPI Acc No: 2002-005933/ 200201
XRPX Acc No: N02-005023

Program converter for multi-thread type microprocessor, converts
conditional branch in middle of program into parallel unconditional
branch using FORK command, when thread execution devices operate
parallelly

Patent Assignee: NEC CORP (NIDE)
Number of Countries: 001 Number of Patents: 001
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week
JP 2001282549	A	20011012	JP 200093508	A	20000330	200201 B

Priority Applications (No Type Date): JP 200093508 A 20000330
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
JP 2001282549	A		27	G06F-009/45	

Abstract (Basic): JP 2001282549 A

NOVELTY - The converter converts conditional branch in the middle
of a program into parallel unconditional branch using FORK command,
when thread execution devices operate parallelly. A command
rearrangement unit (23) rearranges the commands to be executed before
and after of the FORK command such that FORK command is executed
earlier.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are also included for the
following:

(a) Program conversion method;
(b) Recording medium storing program for executing program
conversion method
USE - For information processor such as multi-thread type
microprocessor.

ADVANTAGE - Enables performing parallel execution of several
threads on the middle of the program exactly, using FORK command and
hence workability is enhanced.

DESCRIPTION OF DRAWING(S) - The figure shows the internal component
of the program converter. (Drawing includes non-English language text).

Command rearrangement unit (23)
pp; 27 DwgNo 2/21

Title Terms: PROGRAM; CONVERTER; MULTI; THREAD; TYPE; MICROPROCESSOR;
CONVERT; CONDITION; BRANCH; MIDDLE; PROGRAM; PARALLEL; UNCONDITIONAL;
BRANCH; FORK; COMMAND; THREAD; EXECUTE; DEVICE; OPERATE; PARALLEL

Derwent Class: T01

International Patent Class (Main): G06F-009/45

International Patent Class (Additional): G06F-009/34; G06F-009/38;
G06F-009/46

File Segment: EPI

4/5/2 (Item 1 from file: 347)
DIALOG(R) File 347: JAPIO
(c) 2004 JPO & JAPIO. All rts. reserv.

07054914 **Image available**
DEVICE AND METHOD FOR CONVERTING PROGRAM AND RECORDING MEDIUM

PUB. NO.: 2001-282549 A]
PUBLISHED: October 12, 2001 (20011012)
INVENTOR(s): SAKAI JUNJI
APPLICANT(s): NEC CORP

APPL. NO.: 2000-093508 [JP 200093508]
FILED: March 30, 2000 (20000330)
INTL CLASS: G06F-009/45; G06F-009/34; G06F-009/38; G06F-009/46

ABSTRACT

PROBLEM TO BE SOLVED: To provide a program conversion device performing paralleling at an intermediate program level for a multithread microprocessor.

SOLUTION: A paralleling device 11 is provided with a FORK part deciding part 21, a register allocation part 22 and an instruction rearranging part 23. The FORK part deciding part 21 decides a FORK part and a FORK system based on a result obtained by trying register allocation in the register allocation part 22, the number of memory data depending parts and branch probability and data dependency generation frequency, which are obtained from a profile information file 5, with respect to the inputted intermediate program. The instruction rearranging part 23 rearranges instructions before and after a FORK instruction following the decision.

COPYRIGHT: (C)2001,JPO

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-282549

(P2001-282549A)

(43) 公開日 平成13年10月12日 (2001.10.12)

(51) Int.Cl. ⁷	識別記号	F I	テーマコード [*] (参考)
G 0 6 F	9/45	G 0 6 F	3 3 0 5 B 0 1 3
	9/34		3 3 0 K 5 B 0 3 3
	9/38		3 6 0 B 5 B 0 8 1
	9/46		3 2 2 F 5 B 0 9 8

審査請求 有 請求項の数33 O L (全 27 頁)

(21) 出願番号 特願2000-93508 (P2000-93508)

(22) 出願日 平成12年3月30日 (2000.3.30)

(71) 出願人 000004237

日本電気株式会社

東京都港区芝五丁目7番1号

(72) 発明者 酒井 淳嗣

東京都港区芝五丁目7番1号 日本電気株式会社社内

(74) 代理人 100080816

弁理士 加藤 朝道

Fターム(参考) 5B013 BB18

5B033 AA14 BE00

5B081 CC23 CC25 CC32

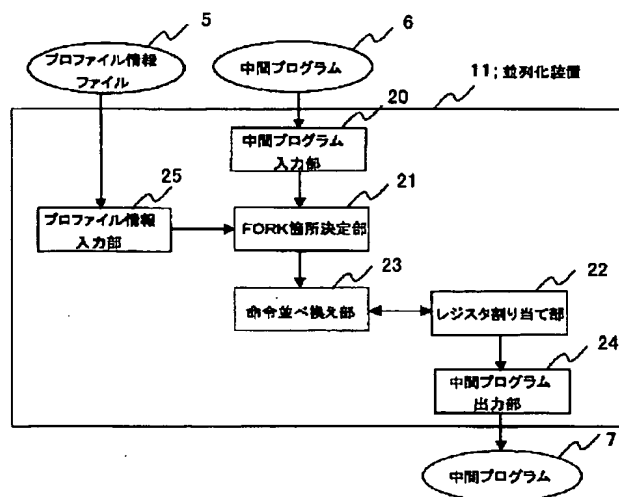
5B098 AA10 GA05 GC14

(54) 【発明の名称】 プログラム変換装置及び方法並びに記録媒体

(57) 【要約】

【課題】 中間プログラムレベルでマルチスレッドマイクロプロセッサ向けの並列化を行うプログラム変換装置の提供。

【解決手段】 並列化装置11は、FORK箇所決定部21とレジスタ割り当て部22と命令並べ換え部23を備え、入力した中間プログラムに対し、FORK箇所決定部21はレジスタ割り当て部22でレジスタ割り当てを試行した結果とメモリデータ依存箇所の数とプロフィール情報ファイル5から得た分岐確率及びデータ依存発生頻度をもとに、FORK箇所と、FORK方式を決定し、命令並べ換え部23は、その決定に沿ってFORK命令前後の命令を並べ換える。



【特許請求の範囲】

【請求項 1】複数のプログラムカウンタと、複数のスレッド実行装置と、を備え、前記複数のスレッド実行装置は、前記複数のプログラムカウンタに従って、複数のスレッドの命令を、同時に、フェッチ、解釈、実行し、スレッド生成時以降に、レジスタセットに及ぼした変更を、後に取り消し可能な制御投機的モードでスレッドを実行すること、及び、

自スレッドがメモリ領域から値をロードした後に、自スレッドを生成した親スレッドが同一メモリ領域に値をストアした場合に、自スレッドの少なくとも当該ロード以降の処理結果を破棄し、それらの処理を再実行するデータ依存投機的モードでスレッドを実行すること、が可能とされ、

命令セットとして、

前記スレッド実行装置で実行中のスレッドが制御投機的モードの新たなスレッドを生成すること、

指定された条件が成立していれば自スレッドを終了させると共に、自スレッドが生成した制御投機的モードのスレッドの制御投機的モードを解除すること、

生成した制御投機的モードのスレッドを破棄すること、自スレッドが生成するスレッドが指定されたアドレスのメモリ領域からのロードを行う際に、その動作を一時停止させることをあらかじめ指示すること、

指定されたメモリアドレスに対する前記ロード一時停止指示を解除すること、前記スレッド実行装置で実行中のスレッドがデータ依存投機的モードの新たなスレッドを生成すること、及び、

自スレッドが生成したデータ依存投機的モードのスレッドのデータ依存投機的モードを解除すること、

が、単一又は高々数個の機械語命令の組合せで実行できる命令セットを有するマルチスレッドプロセッサに対して、与えられた原始プログラムを前記マルチスレッドプロセッサ向けに変換するプログラム変換装置であって、並列化に先立ってレジスタ割り当てを試み、中間プログラム上の各変数、及び中間項のレジスタ割り当て状況を予測するレジスタ割り当て試行部と、

前記レジスタ割り当て試行部でのレジスタ割り当て試行結果に基づいて、前記中間プログラムにおける条件分岐部分をスレッド生成命令を用いた並列コードに変換するか否か決定し、及び、並列コードでの並列実行方式の決定を行うフォーク（FORK）箇所決定部と、

前記フォーク（FORK）箇所決定部での決定結果に基づいて、前記中間プログラム中の条件分岐部分を、スレッド生成命令を用いた並列コードに変換し、前記レジスタ割り当て試行結果を参照して、スレッド間のメモリを介したデータ依存関係を保証する命令を、前記スレッド生成命令の前後に挿入すると共に、スレッド生成が早い段階で行われるように、前記スレッド生成命令の前後の命令を並べ換える命令並べ換え部と、

並列化され並べ換えられた命令列に対して、物理レジスタが割り当てられるか否かに関して、前記レジスタ割り当ての試行時と同じ割り当て結果となるように確定的なレジスタ割り当てを行うレジスタ割り当て部と、を備える、ことを特徴とするプログラム変換装置。

【請求項 2】前記フォーク（FORK）箇所決定部が、現在処理対象としている中間プログラム中の基本ブロックから、該基本ブロック末尾にある条件分岐命令の分岐先基本ブロック各々へのメモリを介したデータ依存関係を調査し、

分岐先の各々について、分岐先基本ブロック中であってデータ依存を引き起こしているメモリ参照命令のうち、最も先頭にある命令の、該分岐先基本ブロック先頭からの命令ステップ数を数え、

前記命令ステップ数が大きな側の分岐先基本ブロックを、並列実行させる新たなスレッドとして選択する、ことを特徴とする請求項 1 に記載のプログラム変換装置。

【請求項 3】前記フォーク（FORK）箇所決定部が、分岐先基本ブロックにてメモリを介したデータ依存命令の位置を求める際、前記命令ステップ数に代えて、各命令の推定実行サイクル数を積算した値を用いる、ことを特徴とする請求項 2 に記載のプログラム変換装置。

【請求項 4】請求項 1、請求項 2 または請求項 3 に記載のプログラム変換装置において、

初回到原始プログラムから目的プログラムに変換する際に、前記プログラム変換装置内の中間プログラムでの基本ブロックと、出力する目的プログラムでの機械命令アドレスとの対応をとるためのアドレス対応情報を、前記目的プログラムと併せて出力し、

目的プログラム実行装置が、前記目的プログラムと前記アドレス対応情報を読み込んで、前記目的プログラムを実行すると共に、前記目的プログラムの実行時の基本ブロック間の分岐プロファイル情報と、基本ブロック間でメモリを介して発生したデータ依存情報とを含むプロファイル情報を出力し、

次に、前記プログラム変換装置が、原始プログラムを並列化して目的プログラムに変換する際に、前記フォーク（FORK）箇所決定部が、前記プロファイル情報を参照して、条件分岐において、制御の流れる確率が高い分岐先基本ブロック及び条件分岐において、データ依存が発生する確率が低い分岐先基本ブロックを並列実行する新スレッドとして優先的に選択する、ことを特徴とする、プログラム変換装置。

【請求項 5】前記フォーク（FORK）箇所決定部が、並列実行する新スレッドの実行開始点として選択した条件分岐先基本ブロックに対し、中間プログラムでのメモリを介したデータ依存を解析した結果と、前記プロファイル情報から得たデータ依存発生確率とに基づき、データ依存を引き起こす異なるメモリアドレスの箇所の数が定められた数より少ない場合には、新スレッドが該メモ

3

リアドレスからロードする動作を一時停止させるように命令を生成し、

一方、データ依存を引き起こす異なるメモリアドレスの箇所の数が定められた数より多い場合には、データ依存発生確率が定められた確率より低いかな否かを調べ、確率が低い場合には、データ依存投機的モードで新スレッドを生成するように命令を生成し、
確率が高い場合には、当該箇所での並列化変換を取りやめるように制御する、ことを特徴とする請求項 4 に記載のプログラム変換装置。

【請求項 6】前記フォーク (FORK) 箇所決定部が、現在処理対象としている中間プログラム中の基本ブロックから、該基本ブロック末尾にある条件分岐命令の分岐先基本ブロック各々へのメモリを介したデータ依存関係を調査し、該調査したデータ依存関係と、前記プロファイル情報から得た条件分岐確率とを総合した結果、当該条件分岐での分岐先基本ブロック各々に関して、分岐確率の間に定められた以上の差異がなく、且つ、メモリを介したデータ依存発生時期の早遅にも定められた以上の差異がない場合に、当該条件分岐部分を並列化しないように決定する、ことを特徴とする請求項 4 または請求項 5 に記載のプログラム変換装置。

【請求項 7】原始 (ソース) プログラムを構文解析して中間プログラムを生成する構文解析部と、
前記中間プログラムに対して並列化を含む最適化処理を行う並列化部と、

前記並列化装置で最適化済みの中間プログラムからターゲットプロセッサ装置向けの命令コードからなる目的プログラム (オブジェクトコード) を生成出力するコード生成部と、

を備えたプログラム変換装置において、

前記並列化部が、

前記中間プログラムを読み込んで制御フローやデータフローを解析する中間プログラム入力部と、

並列化に先立ってレジスタ割り当てを試み、中間プログラム上の各変数、及び中間項のレジスタ割り当て状況を予測するとともに、レジスタの割り当てを実行するレジスタ割り当て部と、

前記レジスタ割り当て試行結果に基づいて、前記中間プログラムにおける条件分岐部分をスレッド生成命令を用いた並列コードに変換する箇所を決定するフォーク (FORK) 箇所決定部と、

前記 FORK 箇所決定部で決定された並列化箇所とデータフローなどの情報から、前記並列化箇所前後の命令の並べ換えを行う命令並べ換え部と、

前記並列化を含む変換を終えた命令列を再度中間プログラム形式で出力する中間プログラム出力部と、

を備えた、ことを特徴とするプログラム変換装置。

【請求項 8】前記並列化部が、前記ターゲットプロセッサ装置で前記目的プログラムを実行して出力されるプロ

4

ファイル情報を入力して内部形式に変換するプロファイル情報入力部を備え、

前記フォーク (FORK) 箇所決定部が、前記レジスタ割り当て試行結果と、前記プロファイル情報に基づき、前記中間プログラムにおける条件分岐部分をスレッド生成命令を用いた並列コードに変換する箇所を決定するとともに、並列コードによる並列実行方式を決定する、ことを特徴とする請求項 7 記載のプログラム変換装置。

【請求項 9】前記命令並べ換え部が、前記フォーク (FORK) 箇所決定部での決定結果に基づいて、前記中間プログラム中の条件分岐部分を、スレッド生成命令を用いた並列コードに変換し、前記レジスタ割り当て試行結果を参照して、スレッド間のメモリを介したデータ依存関係を保証する命令を、前記スレッド生成命令の前後に挿入すると共に、スレッド生成が早い段階で行われるように、前記スレッド生成命令の前後の命令を並べ換える、ことを特徴とする請求項 7 記載のプログラム変換装置。

【請求項 10】前記レジスタ割り当て部が、並列化され並べ換えられた命令列に対して、物理レジスタが割り当てられるか否かに関して前記レジスタ割り当ての試行時と同じ割り当て結果となるように確定的なレジスタ割り当てを行う、ことを特徴とする請求項 7 記載のプログラム変換装置。

【請求項 11】原始プログラムを構文解析部で構文解析して得られる中間プログラムに対して、ターゲットプロセッサ装置向けに、並列化を含む最適化処理を行うプログラム変換装置において、

前記中間プログラム上で、前記ターゲットプロセッサ装置でのレジスタの割り当てを試行し、レジスタ割り当て情報を実際の割り当てに先行して取得するレジスタ割り当て試行手段と、

前記中間プログラムに対して、前記ターゲットプロセッサ装置でのメモリを介して発生するデータ依存の距離の計算を行う手段と、

前記中間プログラム上で、前記メモリを介するデータ依存の距離を考慮して、フォーク (FORK) 先を決定し、条件分岐をスレッド生成命令に置き換える手段と、
前記レジスタ割り当て試行結果を参照して、前記中間プログラム上で、前記スレッド生成命令前後の命令の並べ換えを行う手段と、

を備えたことを特徴とするプログラム変換装置。

【請求項 12】前記条件分岐をスレッド生成命令に置き換える手段が、

前記条件分岐の 2 つの分岐先の各々に対して、各中間項及び変数のデータ依存の距離の最小値を計算する手段と、

前記条件分岐の 2 つの分岐に関して、それぞれ求めた 2 つのデータ依存の距離の最小値を比較し、両者に所定値以上の差がある場合、前記データ依存の距離の最小値の

5

大きい側の分岐方向をフォーク（FORK）先とし、該条件分岐箇所をフォーク（FORK）箇所に選定し、一方、前記データ依存の距離の最小値に前記所定値以上の差がない場合には、元の中間プログラム内で分岐先であった側をフォーク（FORK）先とし、該条件分岐箇所をフォーク（FORK）箇所候補に選定する手段と、を備えた、ことを特徴とする請求項 11 記載のプログラム変換装置。

【請求項 13】請求項 11 記載のプログラム変換装置において、

前記プログラム変換装置から出力される目的プログラムを実行するプロセッサ装置から出力されるプロファイル情報を入力し、前記プロファイル情報から、条件分岐確率及びデータ依存発生頻度を計算する手段と、前記データ依存の距離と、前記条件分岐確率及び前記データ依存発生頻度と、データ依存を引き起こす互いに異なるメモリアドレスの箇所の数から、フォーク（FORK）先及びデータ依存保証方式を決定し、条件分岐をスレッド生成命令に置き換える手段と、を備えたことを特徴とするプログラム変換装置。

【請求項 14】前記ターゲットプロセッサ装置が、複数のプログラムカウンタと、複数のスレッド実行装置と、を備え、前記複数のスレッド実行装置は、前記複数のプログラムカウンタに従って、複数のスレッドの命令を、同時に、フェッチ、解釈、実行し、

スレッド生成時以降に、レジスタセットに及ぼした変更を、後に取り消し可能な制御投機的モードでスレッドを実行すること、及び、

自スレッドがメモリ領域から値をロードした後に、自スレッドを生成した親スレッドが同一メモリ領域に値をストアした場合に、自スレッドの少なくとも当該ロード以降の処理結果を破棄し、それらの処理を再実行するデータ依存投機的モードでスレッドを実行すること、が可能とされ、

命令セットとして、

前記スレッド実行装置で実行中のスレッドが制御投機的モードの新たなスレッドを生成すること、

指定された条件が成立していれば自スレッドを終了させると共に、自スレッドが生成した制御投機的モードのスレッドの制御投機的モードを解除すること、

生成した制御投機的モードのスレッドを破棄すること、自スレッドが生成するスレッドが指定されたアドレスのメモリ領域からのロードを行う際に、その動作を一時停止させることをあらかじめ指示すること、

指定されたメモリアドレスに対する前記ロード一時停止指示を解除すること、

前記スレッド実行装置で実行中のスレッドがデータ依存投機的モードの新たなスレッドを生成すること、及び、自スレッドが生成したデータ依存投機的モードのスレッドのデータ依存投機的モードを解除すること、

6

が、単一又は高々数個の機械語命令の組合せで実行できる命令セットを有するマルチスレッドプロセッサよりなる、ことを特徴とする請求項 7 乃至 13 のいずれか一に記載のプログラム変換装置。

【請求項 15】原始プログラムをコンパイルしてマルチスレッド型のターゲットプロセッサ装置向けの目的プログラムを出力するプログラム変換装置において、構文解析の結果出力される中間プログラムに対して並列化を含む最適化処理を行う方法であって、

10 (a) 並列化に先立ってレジスタ割り当てを試み、中間プログラム上の各変数、及び中間項のレジスタ割り当て状況を予測するレジスタ割り当て試行ステップと、

(b) 前記レジスタ割り当て試行結果に基づいて、前記中間プログラムにおける条件分岐部分をスレッド生成命令を用いた並列コードに変換するか否かの決定を行うか、もしくは、並列コードに変換するか否かの決定と並列コードに変換する場合その並列実行方式の決定を行うフォーク（FORK）箇所決定ステップと、

20 (c) 前記フォーク（FORK）箇所決定ステップでの決定結果に基づいて、前記中間プログラム中の条件分岐部分を、スレッド生成命令を用いた並列コードに変換し、前記レジスタ割り当て試行結果を参照して、スレッド間のメモリを介したデータ依存関係を保証する命令を、前記スレッド生成命令の前後に挿入すると共に、スレッド生成が早い段階で行われるように、前記スレッド生成命令の前後の命令を並べ換える命令並べ換えステップと、

30 (d) 並列化され並べ換えられた命令列に対して前記レジスタ割り当ての試行時と同じ割り当て結果となるように確定的なレジスタ割り当てを行うレジスタ割り当てステップと、

を含む、ことを特徴とするプログラム並列化方法。

【請求項 16】前記フォーク（FORK）箇所決定ステップが、前記条件分岐の 2 つの分岐先の各々に対して、各中間項及び変数のデータ依存の距離の最小値を計算し、前記条件分岐の 2 つの分岐に関して、それぞれ求めた 2 つのデータ依存の距離の最小値を比較し、両者に所定値以上の差がある場合、前記データ依存の距離の最小値の大きい側の分岐方向をフォーク（FORK）先と

40 し、該条件分岐箇所をフォーク（FORK）箇所に選定し、一方、前記データ依存の距離の最小値に前記所定値以上の差がない場合には、元の中間プログラム内で分岐先であった側をフォーク（FORK）先とし、該条件分岐箇所をフォーク（FORK）箇所候補に選定する、ことを特徴とする請求項 15 記載のプログラム並列化方法。

【請求項 17】前記フォーク（FORK）箇所決定ステップが、現在処理対象としている中間プログラム中の、分岐や合流のない一続きのブロック（「基本ブロック」という）から、該基本ブロック末尾にある条件分岐命令

の分岐先基本ブロック各々へのメモリを介したデータ依存関係を調査し、

分岐先の各々について、分岐先基本ブロック中においてデータ依存を引き起こしているメモリ参照命令のうち、最も先頭にある命令の、該分岐先基本ブロック先頭からの命令ステップ数を数え、

前記命令ステップ数が大きな側の分岐先基本ブロックを、並列実行させる新たなスレッドとして選択する、ことを特徴とする請求項 15 に記載のプログラム並列化方法。

【請求項 18】前記フォーク (FORK) 箇所決定ステップが、分岐先基本ブロックにてメモリを介したデータ依存命令の位置を求める際、前記命令ステップ数に代えて、各命令の推定実行サイクル数を積算した値を用いる、ことを特徴とする請求項 17 に記載のプログラム並列化方法。

【請求項 19】前記プログラム変換装置で、まず、原始プログラムから目的プログラムに変換する際に、中間プログラムでの基本ブロックと、出力する目的プログラムでの機械命令アドレスとの対応をとるためのアドレス対応情報を、前記目的プログラムと併せて出力し、前記目的プログラムを実行するプロセッサ装置が、前記目的プログラムと前記アドレス対応情報を読み込んで、前記目的プログラムを実行すると共に、前記目的プログラムの実行時の基本ブロック間の分岐プロファイル情報と、基本ブロック間でメモリを介して発生したデータ依存情報とを含むプロファイル情報を出力し、次に、前記プログラム変換装置が、原始プログラムを並列化して、目的プログラムに変換する際に、前記フォーク (FORK) 箇所決定ステップにおいて、前記プロファイル情報を参照して、条件分岐において、制御の流れる確率が高い分岐先基本ブロック及び条件分岐において、データ依存が発生する確率が低い分岐先基本ブロックを並列実行する新スレッドとして優先的に選択する、ことを特徴とする、請求項 15 に記載のプログラム並列化方法。

【請求項 20】前記フォーク (FORK) 箇所決定ステップが、並列実行する新スレッドの実行開始点として選択した条件分岐先基本ブロックに対し、中間プログラムでのメモリを介したデータ依存を解析した結果と、前記プロファイル情報から得たデータ依存発生確率とに基づき、データ依存を引き起こす異なるメモリアドレスの箇所の数が定められた数よりも少ない場合には、新スレッドが該メモリアドレスからロードする動作を一時停止させるように命令を生成し、一方、データ依存を引き起こす異なるメモリアドレスの箇所の数が定められた数より多い場合には、データ依存発生確率が定められた確率より低いかなかを調べ、確率が低い場合には、データ依存投機的モードで新スレッドを生成するように命令を生成し、

確率が高い場合には、当該箇所での並列化変換を取りやめるように制御する、ことを特徴とする、請求項 19 に記載のプログラム並列化方法。

【請求項 21】前記フォーク (FORK) 箇所決定ステップが、現在処理対象としている中間プログラム中の基本ブロックから、該基本ブロック末尾にある条件分岐命令の分岐先基本ブロック各々へのメモリを介したデータ依存関係を調査し、該調査したデータ依存関係と、前記プロファイル情報から得た条件分岐確率とを総合した結果、当該条件分岐での分岐先基本ブロック各々に関して、分岐確率の間に、予め定められた以上の差異がなく、且つ、メモリを介したデータ依存発生時期の早遅にも、予め定められた以上の差異がない場合に、当該条件分岐部分を並列化しないように決定する、ことを特徴とする請求項 19 に記載のプログラム並列化方法。

【請求項 22】前記フォーク (FORK) 箇所決定ステップが、

(1) 条件分岐命令が前記中間プログラム中のループ構造の戻り枝に相当する分岐命令であるかなかを判定するステップと、

(2) 前記条件分岐命令がループ戻り枝に相当する分岐命令である場合、当該戻り枝方向、即ちループ継続方向を FORK 先として、この条件分岐箇所を、FORK 箇所に選定するステップと、

(3) 前記条件分岐命令が、ループ戻り枝分岐ではない場合、条件分岐の 2 つの分岐先の各々に対して、各中間項／変数のデータ依存の距離の最小値を計算するステップと、

(4) 前記条件分岐の 2 つの分岐に関してそれぞれ求めた 2 つのデータ依存の距離の最小値を比較し、両者に所定値以上の差があるかなかを判定するステップと、

(5) 前記 2 つのデータ依存の距離の最小値に前記所定値以上の差がある場合、データ依存の距離の最小値の大きい側の分岐方向を FORK 先とし、この条件分岐箇所を、FORK 箇所に選定するステップと、

(6) データ依存の距離の最小値に前記所定値以上の差がない場合、元の間プログラム内で分岐先であった側 (分岐命令の taken 側) を FORK 先とし、この条件分岐箇所を FORK 候補に選定するステップと、を含むことを特徴とする請求項 15 に記載のプログラム並列化方法。

【請求項 23】前記データ依存の距離が、現在、処理対象としている基本ブロック内で定義され、その分岐先で参照される可能性がある中間項及び変数のうち、メモリ上に配置されると見込まれる中間項、及び変数の各々に対して、分岐先の基本ブロックの中で、該メモリ参照命令が、先頭からどれぐらいの位置にあるかを、中間プログラム中のステップ数で表したものである、ことを特徴とする請求項 22 に記載のプログラム並列化方法。

【請求項 24】前記データ依存の距離を求める際、各命

令が目的アーキテクチャのプロセッサ上で実行される際に要すると推定されるサイクル数を用いる、ことを特徴とする請求項 2 に記載のプログラム並列化方法。

【請求項 25】前記命令並べ換えステップが、

(1) 前記中間プログラム中の各中間項及び変数がレジスタに対応付けられるかメモリに対応付けられるかレジスタの割り当て状況を調べるステップと、

(2) 現在処理対象としている基本ブロックの末尾にある分岐命令を、制御投機モード FORK 命令に置換し、その際、制御投機モード FORK 命令のオペランドである FORK 先は、前記 FORK 箇所決定ステップで選択された FORK 先とするステップと、

(3) 前記中間プログラム中で、制御投機モード FORK 命令の直前にある分岐条件式を、制御投機モード FORK 命令直後に移動させると共に、該移動先の直後すなわち当該基本ブロックの末尾に、分岐条件成立時には自スレッドを終了させて子スレッドを確定モードすなわち非制御投機モードに移行させ、分岐条件非成立時には子スレッドを破棄させて自スレッドが後続命令列の実行を続ける、一群の命令列を、挿入するステップと、

(4) 現在処理対象としている基本ブロックにおいて、前記制御投機モード FORK 命令よりも手前、即ち、上流側にある命令文のうち、メモリに対応付けられた中間項及び変数への代入となる文の各々について、その代入文を、前記制御投機モード FORK 命令よりも後方、即ち、下流側へ移動させると共に、前記制御投機モード FORK 命令の直前に、ブロック設定命令を挿入し、代入文の移動先の直後に、ブロック解除命令を挿入するステップと、

(5) 前記ステップ (2) の FORK 変換処理で仮定したレジスタ割り当て状況に、レジスタを割り当てるように指示するステップと、
を含む、ことを特徴とする請求項 15 に記載のプログラム並列化方法。

【請求項 26】前記フォーク (FORK) 箇所決定ステップが、

(1) 条件分岐命令が前記中間プログラム中のループ構造の戻り枝に相当する分岐命令であるか否かを判定するステップと、

(2) ループ構造の戻り枝に相当する場合、前記戻り枝方向を FORK 先として仮決定するステップと、

(3) 入力したプロファイル情報に基づいて、当該条件分岐命令の taken (分岐成立) 側と、fall through (フォールスルー) 側が選択される確率と、を計算するステップと、

(4) 前記計算された 2 つ分岐の確率間に所定の基準値以上の違いがあるかどうかを判定するステップと、

(5) 前記 2 つ分岐の確率の違いが前記基準値を越えていれば、確率の高い側を、FORK 先として仮決定するステップと、

(6) 条件分岐の 2 つの分岐先各々に対して、データ依存の距離の最小値を計算するステップと、

(7) 条件分岐の 2 つの分岐に関して求めた 2 つのデータ依存の距離の最小値を比較し、両者に所定値以上の差があるか否かを判定するステップと、

(8) 2 つのデータ依存の距離の最小値に前記所定値以上の差があるか又はデータ依存がない場合、データ依存の距離の最小値の大きい側の分岐方向を FORK 先として決定するステップと、

10 (9) 前記ステップ (2) 又は (5) で仮決定された FORK 先に対して、データ依存の距離の最小値を計算し、仮決定した FORK 先側のデータ依存の距離の最小値が所定値以上あるか否かを判定するステップと、

(10) 仮決定した FORK 先側のデータ依存の距離の最小値に所定値以上の差があるか、メモリを介したデータ依存がなければ、ステップ (2) 又はステップ (5) で仮決定した FORK 先を、正式な FORK 先として確定するステップと、

20 (11) 前記データの依存の距離の最小値が一定水準に満たないと判断された場合には、当該基本ブロックを、FORK 箇所から除外するステップと、

(12) 入力したプロファイル情報からデータ依存発生頻度を計算するステップと、

(13) データの依存発生頻度が一定水準より高いか否かを判断し、低い場合、前記中間プログラム中で、FORK 元基本ブロックから、FORK 先基本ブロックへのデータ依存を引き起こし得るメモリ上の中間項／変数の個数を数え挙げ、それが一定水準より多いか否かを判定し、データ依存箇所数が一定水準より多ければ DSP 方式による FORK を用い、そうでなければ BLOCK 方式による FORK を用いることとし、その情報を、中間プログラム中の FORK 命令に付与するステップと、

(14) データの依存発生頻度が一定水準より高い場合、データ依存しているメモリ上の変数の個数を数え挙げ、その数が、一定水準より少なければ、BLOCK 方式による FORK を用い、一定水準より多ければ、当該基本ブロックは FORK 候補から外すステップと、
を含む、ことを特徴とする請求項 15 に記載のプログラム並列化方法。

40 【請求項 27】前記命令並べ換えステップが、

(1) 前記中間プログラム中の各中間項及び変数がレジスタに対応付けられるかメモリに対応付けられるかレジスタの割り当て状況を調べるステップと、

(2) 現在処理対象としている基本ブロックの末尾にある分岐命令を、制御投機モード FORK 命令に置換し、その際、制御投機 FORK 命令のオペランドである FORK 先は、前記 FORK 箇所決定ステップで選択された FORK 先とするステップと、

50 (3) 前記中間プログラム中で、制御投機 FORK 命令の直前にある分岐条件式を、制御投機 FORK 命令直後

に移動させると共に、該移動先の直後、即ち当該基本ブロックの末尾に分岐条件成立時には自スレッドを終了させて子スレッドを確定モードすなわち非制御投機モードに移行させ、分岐条件非成立時には、子スレッドを破棄させて自スレッドが後続命令列の実行を続ける一群の命令列を挿入するステップと、

(4) 前記 FORK 箇所決定ステップが決定した当該 FORK 箇所の FORK 時データ保証方式が BLOCK 方式か DSP 方式かをチェックするステップと、

(5) BLOCK 方式の場合、FORK 前のメモリストア文を FORK 後に移動すると共に、必要なブロック設定及びブロック解除命令を挿入し、移動の際には、データ依存関係を検査し、命令実行順序が入れ替わっても演算結果が不変となるもののみ移動させるステップと、

(6) DSP 方式の場合、メモリに対応付けられた中間項への代入文を、FORK 命令後に移動させ、データ依存投機モードで FORK を行うように、前記ステップ

(2) で置換作成された FORK 命令を修正するステップと、

(7) 前記ステップ (2) の FORK 変換処理で仮定したレジスタ割り当て状況をレジスタ割り当てるように指示するステップと、

を含む、ことを特徴とする請求項 15 に記載のプログラム並列化方法。

【請求項 28】原始プログラムをコンパイルしてマルチスレッドプロセッサ装置向けの目的プログラムを生成出力するコンパイラにおいて、構文解析の結果出力される中間プログラムに対して並列化を含む最適化処理であって、

(a) 並列化に先立ってレジスタ割り当てを試み、中間プログラム上の各変数、及び中間項のレジスタ割り当て状況を予測するレジスタ割り当て試行処理と、

(b) 前記レジスタ割り当て試行結果に基づいて、前記中間プログラムにおける条件分岐部分をスレッド生成命令を用いた並列コードに変換するか否かの決定を行うか、もしくは、並列コードに変換するか否かの決定と並列コードに変換する場合その並列実行方式の決定を行うフォーク (FORK) 箇所決定処理と、

(c) 前記フォーク (FORK) 箇所決定部での決定結果に基づいて、前記中間プログラム中の条件分岐部分を、スレッド生成命令を用いた並列コードに変換し、前記レジスタ割り当て試行結果を参照して、スレッド間のメモリを介したデータ依存関係を保証する命令を、前記スレッド生成命令の前後に挿入すると共に、スレッド生成が早い段階で行われるように、前記スレッド生成命令の前後の命令を並べ換える命令並べ換え処理と、

(d) 並列化され並べ換えられた命令列に対して、物理レジスタが割り当てられるか否かに関して、前記レジスタ割り当ての試行時と同じ割り当て結果となるように確定的なレジスタ割り当てを行うレジスタ割り当て処理

と、

の前記 (a) 乃至 (d) の最適化処理をコンピュータで実行するためのプログラムを記録した記録媒体。

【請求項 29】請求項 28 記載の記録媒体において、前記フォーク (FORK) 箇所決定処理が、現在処理対象としている中間プログラム中の基本ブロックから、該基本ブロック末尾にある条件分岐命令の分岐先基本ブロック各々へのメモリを介したデータ依存関係を調査し、分岐先の各々について、分岐先基本ブロック中にあってデータ依存を引き起こしているメモリ参照命令のうち、最も先頭にある命令の、該分岐先基本ブロック先頭からの命令ステップ数を数え、

前記命令ステップ数が大きな側の分岐先基本ブロックを、並列実行させる新たなスレッドとして選択する、処理を含み、前記処理を、前記コンピュータで実行するためのプログラムを記録した記録媒体。

【請求項 30】請求項 28 記載の記録媒体において、前記フォーク (FORK) 箇所決定処理において、分岐先基本ブロックにてメモリを介したデータ依存命令の位置を求める際、前記命令ステップ数に代えて、各命令の推定実行サイクル数を積算した値を用い処理を、前記コンピュータで実行するためのプログラムを記録した記録媒体。

【請求項 31】請求項 28 記載の記録媒体において、

(f) 前記コンパイラで、まず、原始プログラムから目的プログラムに変換する際に、中間プログラムでの基本ブロックと、出力する目的プログラムでの機械命令アドレスとの対応をとるためのアドレス対応情報を、前記目的プログラムと併せて出力する処理と、

前記目的プログラムを実行するプロセッサ装置が、前記目的プログラムと前記アドレス対応情報を読み込んで、前記目的プログラムを実行すると共に、前記目的プログラムの実行時の基本ブロック間の分岐プロファイル情報と、基本ブロック間でメモリを介して発生したデータ依存情報とを含むプロファイル情報を出力し、

前記コンパイラが、原始プログラムを並列化して、目的プログラムに変換する際に、

(g) 前記フォーク (FORK) 箇所決定処理において、前記プロファイル情報を参照して、条件分岐において、制御の流れる確率が高い分岐先基本ブロック及び条件分岐において、データ依存が発生する確率が低い分岐先基本ブロックを並列実行する新スレッドとして優先的に選択する処理、

の前記 (f) 及び (g) の処理を前記コンピュータで実行するためのプログラムを記録した記録媒体。

【請求項 32】請求項 31 記載の記録媒体において、前記フォーク (FORK) 箇所決定処理が、並列実行する新スレッドの実行開始点として選択した条件分岐先基本ブロックに対し、中間プログラムでのメモリを介したデータ依存を解析した結果と、前記プロファイル情報か

ら得たデータ依存発生確率とに基づき、データ依存を引き起こす異なるメモリアドレスの箇所の数が定められた数より少ない場合には、新スレッドが該メモリアドレスからロードする動作を一時停止させるように命令を生成し、

一方、データ依存を引き起こす異なるメモリアドレスの箇所の数が定められた数より多い場合には、データ依存発生確率が定められた確率より低いかなかを調べ、確率が低い場合には、データ依存投機的モードで新スレッドを生成するように命令を生成し、

確率が高い場合には、当該箇所での並列化変換を取りやめるように制御する、

処理を含み、前記処理を前記コンピュータで実行するためのプログラムを記録した記録媒体。

【請求項 33】請求項 31 記載の記録媒体において、前記フォーク（FORK）箇所決定処理が、現在処理対象としている中間プログラム中の基本ブロックから、該基本ブロック末尾にある条件分岐命令の分岐先基本ブロック各々へのメモリを介したデータ依存関係を調査し、該調査したデータ依存関係と、前記プロファイル情報から得た条件分岐確率とを総合した結果、当該条件分岐での分岐先基本ブロック各々に関して、分岐確率の間に定められた以上の差異がなく、且つ、メモリを介したデータ依存発生時期の早遅にも定められた以上の差異がない場合に、当該条件分岐部分を並列化しないように決定する、処理を含み、前記処理を前記コンピュータで実行するためのプログラムを記録した記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、複数の機械命令を同時に実行できるマルチスレッド型マイクロプロセッサのためのプログラム変換技術に関し、特に、小さな粒度でも実行効率の高い並列プログラムを生成するためのプログラム変換技術に関する。

【0002】

【従来の技術】情報処理装置において、プログラムの実行性能を高めるための一技法として、従来より、プログラムを複数の命令流即ちスレッドに分割し、それらのスレッドを並列実行するマルチスレッド実行技術が採用されている。

【0003】それぞれに実行ユニットを有する複数のプロセッサエレメント（プロセッサ要素）を密に結合したマルチスレッドマイクロプロセッサは、スレッドの生成、同期等の処理を低コストで行えるという特徴を備えており、逐次性が高くスレッド粒度（スレッドのサイズ）が小さいプログラムに対しても、十分な並列処理効果を出すことができる。

【0004】このようなマルチスレッドマイクロプロセッサのアーキテクチャとして、

・文献 1（「On-Chip Microproces

sor 指向制御並列アーキテクチャ MUSCAT の提案」（鳥居他、並列処理シンポジウム JSP97 論文集、情報処理学会、第 229-236 頁、May 1997））、

・文献 2（「オンチップ制御並列プロセッサ MUSCAT の提案」（鳥居他、情報処理学会論文誌、Vol. 39, No. 6, June 1998））、

・文献 3（特開平 10-78880 号公報）

等に表示されたアーキテクチャが知られている。

10 【0005】まず、前記文献 2 に記載されている MUSCAT アーキテクチャについて説明する。MUSCAT は、複数のプロセッサエレメントを 1 チップに集積し、「制御並列」と呼ぶマルチスレッド処理を効率良く実行するアーキテクチャである。各プロセッサエレメントは、それぞれ独立した演算器と独立したレジスタセットを持つ。他方、各プロセッサエレメントはメモリ空間を共有する。

20 【0006】MUSCAT における制御並列実行方式について説明する。図 18 に示すように、MUSCAT アーキテクチャのプロセッサは、スレッド生成のための FORK（フォーク）命令を持ち、この機械命令 1 つで、隣接プロセッサエレメントに新しいスレッドを生成することができる。

30 【0007】MUSCAT は、1 つのスレッドが FORK 命令を実行できる回数を高々 1 回に制限する「FORK 1 回モデル」を採用している。各プロセッサエレメントは、単方向のリング状に結合され、あるプロセッサエレメントで実行中のスレッドが FORK 命令を実行すると、下流側に隣接するプロセッサエレメントに新しいスレッドが生成される。スレッドが TERM（ターム）命令（スレッド終了命令）を実行すると、自スレッドが終了する。

40 【0008】プロセッサエレメント間のデータのやりとりは、レジスタまたはメモリを通じて行われる。FORK 命令を実行すると、当該スレッド（「親スレッド」と呼ぶ）を実行しているプロセッサエレメントのレジスタセットの内容が、生成された新スレッド（「子スレッド」と呼ぶ）を実行するプロセッサエレメントのレジスタセットに論理的にコピーされる。また、親スレッドが FORK 命令実行前にメモリにストアした値は、子スレッドがロードして参照することができる。

【0009】親スレッドが子スレッドを生成した後、親スレッドが子スレッドにデータを渡す唯一の方法は、メモリを介してデータを渡す、というものである。この場合、親スレッドのストアと子スレッドのロードの実行順序を正しいものにするために、一種の同期をとる必要がある。この同期のための措置を、「データ依存保証」と呼ぶ。

50 【0010】MUSCAT は、上記データ依存保証のための方式として、

・ 予め同期を明示的に宣言する BLOCK 方式、
 ・ 同期無しで実行して、もし同期に失敗したら実行をやり直す DSP 方式、

の 2 種類のデータ依存保証方式を備える。

【0011】図 19 は、BLOCK 方式の動作を説明する図である。BLOCK 方式によるデータ依存保証方式は、MUSCAT が備える BLOCK (ブロック) 命令と、RELEASE (リリース) 命令を用いる方式である。

【0012】親スレッドは、ストア対象メモリアドレスを BLOCK 命令で指定してから FORK 命令を実行し、メモリストアを行う。その後、親スレッドは、RELEASE 命令を実行して、当該メモリアドレスにデータが準備できたことを表明する。親スレッドが BLOCK したアドレスからのロードを子スレッドが実行しようとする、親スレッドが RELEASE 命令を実行するまで、子スレッドのロード命令は完了しない。このようにして、親スレッドがメモリに書き込んだ値 (メモリにストアした値) を、子スレッドが正しく読み出す (メモリからロードする) ことができる。他のデータ依存保証方式である DSP 方式は、投機実行方式の一つであり、詳細は後述する。

【0013】次に、図 20 を参照して、MUSCAT アーキテクチャの投機実行機能について説明する。MUSCAT の SPFORK 命令は FORK 命令の一種であり、制御投機モードのスレッド生成を指示する。SPFORK 命令で生成された子スレッドは、いつでも実行を取り消せる状態で実行される。即ち、レジスタ値の変更は許されるが、外部メモリへのアクセスは抑止される。

【0014】図 20 (A) に示すように、親スレッドが THFIX 命令 (スレッド確定命令) を実行すると、子スレッドの制御投機モードは解除され、内部で溜っていたメモリ変更操作が実際にメモリに反映される。

【0015】しかしながら、図 20 (B) に示すように、親スレッドが THABORT 命令 (スレッド破棄命令) を実行すると、子スレッドは破棄され、レジスタやメモリに対して行おうとした変更は全て捨てられる。なお、親スレッドが THFIX 命令、又は THABORT 命令を実行する前に、TERM 命令により終了すると、子スレッドの制御投機モードは解除される。

【0016】制御投機モードのスレッドは、結果的に実行することになる可能性が高いが、現時点では実行してもよいか否か判断がつかない場合等に用いられる。即ち、プログラム内の早い段階で制御投機モードスレッドを生成して並列に処理を行い、後になって生成したスレッドでの処理を本当に行うかどうかを判定してスレッドの確定ないし破棄を行う。

【0017】上記した制御投機モード FORK は、制御の流れを予測し、投機的にスレッド生成するものであるが、これとは独立に、MUSCAT には、データ依存を

予測し、投機的にスレッドを実行する機能が用意されている。このようなスレッド状態は、「データ依存投機モード」と呼ばれる。

【0018】データ依存投機モードを用いる場合、親スレッドは、BLOCK 命令や RELEASE 命令を用いる必要はない。ハードウェアアーキテクチャが、実行時に親子スレッド間のメモリデータ依存 (親スレッドのメモリへのストアと子スレッドのメモリからのロードの時間順序) を検査し、子スレッドが誤った値をロードした場合には、子スレッドを再実行させる。

【0019】前述の BLOCK 方式と並ぶ、もう一つのデータ依存保証方式である DSP 方式は、このデータ依存投機モードを利用する方式である。

【0020】図 21 を参照すると、親スレッドは、データ依存投機モードで FORK することを意味する DSP IN 命令を実行した後で、FORK を行う。これによって生成された子スレッドは、データ依存投機モードで実行を開始する。その後、親スレッドが DSPORT 命令を実行すると、子スレッドはデータ依存投機モードから非データ依存投機モードに戻る。

【0021】データ依存投機モードは、BLOCK/RELEASE 命令が不要であり、データ依存を引き起こすメモリアドレスが事前に分かっている場合でも、FORK を行える、という長所がある。しかしながら、もし、FORK 後にデータ依存が発生すると、子スレッドは誤ったデータに基づいた部分の実行をやり直すことになる、という短所をもつ。

【0022】従って、プログラムに内在するデータ依存の状況によって、BLOCK 方式あるいは DSP 方式のうち、適切な方式を選択すべきである。

【0023】これまで述べた FORK 命令や、データ依存保証方式等の制御並列サポート命令により、MUSCAT アーキテクチャでは、プログラムから並列実行可能な部分をより多く抽出し、プログラムの実行性能を向上させることができる。

【0024】このような MUSCAT アーキテクチャ向けのプログラムは、FORK (フォーク) を行う箇所毎に、前述の命令を用いたコーディングを行わねばならず、このため、高級言語 (高水準プログラミング言語) から機械命令列を自動生成するコンパイラが、従来型のアーキテクチャよりも、一層強く求められる。

【0025】MUSCAT アーキテクチャ向けコンパイラ技術は、文献 4 (「制御並列アーキテクチャ向け自動並列化コンパイル手法」 (酒井他、情報処理学会論文誌、Vol. 40、No. 5、May 1999)) において開示されている。この文献 4 に示された技術の概要を以下に説明する。

【0026】前記文献 4 には、通常の逐次コンパイラが生成した機械命令列を、MUSCAT 向け命令列に変換するトランスレータの仕組みが記載されている。与えら

10

20

30

40

50

れた機械命令列に対して、制御フロー解析及びデータフロー解析を行い、基本ブロック毎にFORK命令を用いた並列化を試みる。ここで、「基本ブロック」とは、途中で制御の分岐や合流が無い、一続きの命令列のことをいう。

【0027】基本ブロック毎の並列化は、まず、基本ブロック末尾の分岐命令を、制御投機モードFORK命令（SPFORK命令）に置き換えることから始まる。これは、ある基本ブロックに後続する複数の基本ブロックの一方を、当該基本ブロックと並列に実行しようとすることを意味する。後続基本ブロックのうちいずれを選択するかは、前記文献4では、ループ構造の場合は、ループの戻り方向がよいとされている他、プロファイル情報を用いて分岐確率の高い方を選択することが示されている。

【0028】次に、当該基本ブロックと、FORK先基本ブロック以降との間でのレジスタ、及びメモリアクセスによるデータ依存を調査する。

【0029】そして、当該基本ブロック内で、FORK命令ができるだけ上流に位置するように、基本ブロック内で命令の並べ換えを行う。命令並べ換えに際しては、データ依存関係に留意し、レジスタを介して正依存がある場合には、その命令をFORK命令よりも上流側に配置する。メモリを介して正依存がある場合には、DSPIN命令ないし依存メモリアドレスを引数としたBLOCK命令を、FORK命令の直前に新たに挿入する。

【0030】両者の使い分けについて、前記文献4では、依存をひき起こすメモリアドレスがストア直前まで決定できない場合、及び、依存をひき起こすメモリアドレスが決定できた場合でもメモリアドレス値がある数より多い場合には、DSPIN命令、それ以外の場合には、BLOCK命令を用いるものとされている。なお、前記文献4では、この他にも、MUSCAT特有の命令生成手順が述べられているが、本発明の主題とは直接の関連性は薄いため、ここでは、あらためて説明は行わない。

【0031】

【発明が解決しようとする課題】ところで、上記した従来の技術は、下記記載の問題点を有している。

【0032】第1の問題点は、FORK先を選択する基準に、まだ改良の余地がある、ことである。

【0033】上記した従来の技術では、静的にはループ構造の把握、動的には実行履歴のプロファイル情報を得て、それに基づいてFORK先を決定している。しかしながら、これらは、制御の流れ上、実行される確率が高い部分をFORKしているにすぎず、データ依存性については、考慮されていない。このため、過度のデータ依存が存在する部分をマルチスレッド実行しようとしても、並列実行区間が短くなったり、あるいはデータ依存箇所待ち合わせが起こるため、結果として、得られる

性能向上は、小さなものになってしまう、という問題がある。

【0034】第2の問題点は、データ依存対処手順がレジスタとメモリとで異なっている、ということである。

【0035】前記文献4等に記載されている手順では、レジスタを介した依存と、メモリを介した依存とで、命令の並べ換えや、新規挿入命令の処理が異なっている。しかしながら、コンパイラ内部では、「中間項」と呼ばれる仮想的な変数を用いた表現が用いられることが一般に行われており、その段階で、レジスタかメモリかの判断に基づく並列化処理を行うことは困難である。

【0036】他方、一般的なコンパイラ内部でも、最終段近くになれば、レジスタ割り当てが済み、レジスタとメモリの区別が確定しているが、この段階で並列化を行おうとすると、既に、最適化された制御フローやデータフローに悪影響を与えないように、プログラム構造を変更する処理が困難になる。

【0037】したがって、本発明は、上記問題点に鑑みてなされたものであって、その目的は、コンパイラ内部の中間項レベルでの確な判断により並列化処理を行えるようにし、もってマルチスレッドマイクロプロセッサ向けプログラム変換装置及び方法並びに記録媒体を提供することである。

【0038】本発明の他の目的は、並列実行時により一層の並列性能を引き出すような目的コードを生成できるプログラム変換装置（コンパイラ）及び方法並びに記録媒体を提供することにある。これ以外の本発明の目的、特徴、利点等は、下記の実施の形態の記載等から、当業者には直ちに明らかとされるであろう。

【0039】

【課題を解決するための手段】前記目的を達成する本発明の第1のプログラム変換装置は、中間プログラム上でレジスタ割り当てを試行して割り当て情報を先行取得するレジスタ割り当てを試行部と、メモリを介して発生するデータ依存の距離の計算部と、を備え、メモリを介するデータ依存の距離を考慮してFORK先を決定し、条件分岐をFORK命令に置き換え、レジスタ割り当て試行結果を参照して、中間プログラム上で、FORK命令前後の命令並べ換えを行う。

【0040】また、本発明の第2のプログラム変換装置は、前記第1のプログラム変換装置に加えて、プロファイル情報から条件分岐確率及びデータ依存発生頻度を計算する部分とを備え、データ依存の距離と条件分岐確率とデータ依存発生頻度とデータ依存メモリ箇所数からFORK先を決定し、条件分岐をFORK命令に置き換え、レジスタ割り当て試行結果を参照して中間プログラム上でFORK命令前後の命令並べ換えを行う。

【0041】本発明の並列化方法は、原始プログラムをコンパイルしてマルチスレッド型のターゲットプロセッサ装置向けの目的プログラムを生成出力するコンパイラ

における、構文解析の結果出力される中間プログラムに対して並列化を含む最適化処理を行う方法であって、

(a) 並列化に先立ってレジスタ割り当てを試み、中間プログラム上の各変数、及び中間項のレジスタ割り当て状況を予測するレジスタ割り当て試行ステップと、

(b) 前記レジスタ割り当て試行結果に基づいて、前記中間プログラムにおける条件分岐部分をスレッド生成命令を用いた並列コードに変換するか否かの決定を行うか、もしくは、並列コードに変換するか否かの決定と並列コードに変換する場合その並列実行方式の決定を行うフォーク (FORK) 箇所決定ステップと、(c) 前記フォーク (FORK) 箇所決定部での決定結果に基づいて、前記中間プログラム中の条件分岐部分を、スレッド生成命令を用いた並列コードに変換し、前記レジスタ割り当て試行結果を参照して、スレッド間のメモリを介したデータ依存関係を保証する命令を、前記スレッド生成命令の前後に挿入すると共に、スレッド生成が早い段階で行われるように、前記スレッド生成命令の前後の命令を並べ換える命令並べ換えステップと、(d) 並列化され並べ換えられた命令列に対して、物理レジスタが割り当てられるか否かに関して、前記レジスタ割り当ての試行時と同じ割り当て結果となるように確定的なレジスタ割り当てを行うレジスタ割り当てステップと、を含む。

【0042】本発明において、ターゲットプロセッサ装置は、好ましくは、複数のプログラムカウンタと、複数のスレッド実行装置と、を備え、前記複数のスレッド実行装置は、前記複数のプログラムカウンタに従って、複数のスレッドの命令を、同時に、フェッチ、解釈、実行し、スレッド生成時以降に、レジスタセットに及ぼした変更を、後に取り消し可能な制御投機的モードでスレッドを実行すること、及び、自スレッドがメモリ領域から値をロードした後に、自スレッドを生成した親スレッドが同一メモリ領域に値をストアした場合に、自スレッドの少なくとも当該ロード以降の処理結果を破棄し、それらの処理を再実行するデータ依存投機的モードでスレッドを実行すること、が可能とされ、命令セットとして、

- ・前記スレッド実行装置で実行中のスレッドが制御投機的モードの新たなスレッドを生成すること、
- ・指定された条件が成立していれば自スレッドを終了させると共に、自スレッドが生成した制御投機的モードのスレッドの制御投機的モードを解除すること、
- ・生成した制御投機的モードのスレッドを破棄すること、
- ・自スレッドが生成するスレッドが指定されたアドレスのメモリ領域からのロードを行う際に、その動作を一時停止させることをあらかじめ指示すること、
- ・指定されたメモリアドレスに対する前記ロード一時停止指示を解除すること、
- ・前記スレッド実行装置で実行中のスレッドがデータ依存投機的モードの新たなスレッドを生成すること、及

び、

- ・自スレッドが生成したデータ依存投機的モードのスレッドのデータ依存投機的モードを解除すること、
- が、単一又は高々数個の機械語命令の組合せで実行できる命令セットを有するマルチスレッドプロセッサよりなる。

【0043】

【発明の実施の形態】次に、本発明の実施の形態について図面を参照して説明する。

10 【0044】【実施の形態1】図1は、本発明の一実施の形態の構成の一例を示す図である。図1を参照すると、プログラム変換装置 (コンパイラ) 2は、原始プログラム (ソースプログラム) 1を読み込んで並列化を含むコンパイル処理を行い、その結果の目的プログラム (ターゲットプログラム) 3を出力する。

【0045】目的プログラム実行装置4は、目的プログラム3を入力して、目的アーキテクチャの命令実行を行うと共に、当該プログラムの実行情報を収集し、プロファイル情報ファイル5を出力する。

20 【0046】図1を参照すると、プログラム変換装置2は、

- ・入力した原始プログラム1の文法 (シンタックス) を解釈して、構文解析し中間プログラムを生成する構文解析装置10と、

- ・構文解析装置10から受け取った中間プログラムに対して並列化を含む最適化処理を行う並列化装置11と、
- ・並列化装置11から受け取った最適化済みの中間プログラムから目的アーキテクチャ用命令列を生成するコード生成装置12と、

30 を備えている。

【0047】並列化装置11は、補助情報として、プロファイル情報ファイル5を入力としてより高度な最適化処理を行うことができる。

【0048】なお、本発明において、構文解析装置10は、周知の構成が用いられ、例えば字句解析 (レキシカルアナリシス)、構文解析 (パーズング)、意味解析 (セマンティックアナリシス) の各解析部からなる構成としてもよいことは勿論である。また最適化済みの中間プログラムから目的アーキテクチャ用命令列を生成するコード生成装置12も、周知のものが用いられる。

40 【0049】図2は、本発明の一実施の形態のプログラムの変換装置2における並列化装置11の構成を示す図である。図1及び図2を参照して、本発明の要部をなす並列化装置11について詳細に説明する。

【0050】並列化装置11は、構文解析装置10が生成した中間プログラム6を受け取り、並列化を含む最適化処理を行い、後段のコード生成装置12に中間プログラム7を渡す。

【0051】並列化装置11は、

- 50 ・入力となる中間プログラム6を読み込んで制御フロー

やデータフローを解析する中間プログラム入力部 20 と、

- ・制御フロー、データフロー及びプロファイル情報をもとに並列化する箇所を決定する FORK 箇所決定部 21 と、
 - ・中間プログラム上の中間項に対してレジスタ割り当てを試行するか、あるいは実行するレジスタ割り当て部 22 と、
 - ・決定された並列化箇所とデータフロー等の情報から並列化箇所前後の命令の並べ換えを行う命令並べ換え部 23 と、
 - ・並列化を含む様々な変換を終えた命令列を再度中間プログラム形式で出力する中間プログラム出力部 24 と、
 - ・一度目的プログラムを実行して得たプロファイル情報ファイル 5 を入力して内部形式に変換するプロファイル情報入力部 25 と、
- を備えている。

【0052】次に、図 3、図 4、及び図 5 のフローチャートを参照して、本発明の一実施の形態の並列化動作について、詳細に説明する。

【0053】本発明の一実施の形態においては、中間プログラム上で、次の制御並列関連命令が記述できる。

【0054】(1) 制御投機 FORK 命令：制御投機 FORK 命令は、オペランドで指し示された命令から実行を開始する制御投機モードの子スレッドを生成し、自スレッドは後続命令の実行を続ける命令である。

【0055】(2) スレッド終了命令：スレッド終了命令は、自スレッドを終了させるとともに子スレッドを確定モードに移行させる命令である。

【0056】(3) 子スレッド破棄命令：子スレッド破棄命令は、制御投機モードの子スレッドを破棄する命令である。

【0057】(4) ブロック設定命令：ブロック設定命令は、オペランドで指し示されたメモリアドレスにブロックを設定し、それによって子スレッドが当該メモリアドレスからのロードを実行しようとした際に子スレッドの実行を一時停止（ブロック）させるように、プロセッサに指示する命令である。

【0058】(5) ブロック解除命令：ブロック解除命令は、オペランドで指し示されたメモリアドレスに設定されているブロックを解除するとともに、当該メモリアドレスからのロードで一時停止（ブロック）されている子スレッドの実行を再開させるように、プロセッサに指示する命令である。

【0059】(6) レジスタ割り当て指示命令：レジスタ割り当て指示命令は、オペランドで指定した中間項ないし変数に物理レジスタを割り当てるように、あるいは逆に、指定した中間項ないし変数にメモリ上の領域を割り当てるように、レジスタ割り当て部に指示する命令である。

【0060】図 3 は、本発明の一実施の形態の FORK 箇所決定部 21 の動作概要を示す図である。FORK 箇所決定部 21 は、与えられた中間プログラム中の各関数単位で、図 3 に示された動作を行う。即ち、

- ・ステップ 25 にて、当該関数内でレジスタ割り当てを試行し、その後、
- ・ステップ 26 にて、当該関数内に含まれる各条件分岐命令に対して FORK 箇所決定処理を行う。

【0061】このステップ 26 の段階では、並列化を含む各種最適化処理は未完了であり、まだ確定的なレジスタ割り当て処理を行う段階ではない。そこで、レジスタ割り当ての途中までを実行し、中間プログラム中のどの中間項／変数にレジスタが割り当てられ、どの中間項／変数がメモリ上に配置されるのか、という情報を得た段階で、実際に割り当ては行わずにレジスタ割り当て処理を中止する。

【0062】レジスタ割り当ての試行は、後述するように、レジスタ割り当て部 22（図 2 参照）で行う。レジスタ割り当て部 22 は、どの中間項／変数に、どのレジスタを割り当てるかを決めた段階で処理を終え、中間項／変数を実際のレジスタに置き換えることなく、レジスタ割り当て状況のみを返す仕組み（機構）を備える。

【0063】図 4 は、各条件分岐命令に対する FORK 箇所決定部 21 の処理（図 3 のステップ 26）の詳細を示す図である。

【0064】図 4 を参照すると、ステップ 31 では、当該条件分岐命令が、入力中間プログラム中のループ構造（繰り返し構造）の戻り枝に相当する分岐命令か否かを判定する。ループ構造の検出方法については、例えば、文献 5（「コンパイラ II 原理・技法・ツール」、A. V. エイホ他著、原田賢一訳、サイエンス社、1990）の第 734 頁～第 737 頁の記載が参照される。

【0065】ステップ 31 の判定で、当該条件分岐命令が、ループ戻り枝に相当する分岐命令である場合、ステップ 35 にて、当該戻り枝方向、即ちループ継続方向を FORK 先として、この条件分岐箇所を、FORK 箇所を選定する。その理由は、一般に、ループ構造は、複数回繰り返し実行される傾向にあるため、ループ戻り枝に相当する分岐命令では、戻り枝側に分岐する確率が高い、からである。

【0066】ステップ 31 において、戻り枝分岐ではない場合、ステップ 32 において、条件分岐の 2 つの分岐先の各々に対してデータ依存の距離の最小値を計算する。

【0067】ここで、「データ依存の距離」とは、現在、処理対象としている基本ブロック内で定義され、その分岐先で参照される可能性がある中間項及び変数のうち、ステップ 26 により、メモリ上に配置されると見込まれる、中間項／変数各々に対して、その分岐先基本ブロックの中で、該メモリ参照命令が先頭から、どれぐら

いの位置にあるかを、中間プログラム中のステップ数で表したものである。

【0068】ステップ32では、2つの分岐先について、各中間項／変数のデータ依存の距離の最小値を計算する。

【0069】次のステップ33では、ステップ32で条件分岐の両側に関して求めた2つのデータ依存の距離の最小値を比較し、両者に一定以上の差があるか否かを判定する。

【0070】ステップ33において、一定以上の差がある場合、ステップ36にて、データ依存の距離の最小値の大きい側の分岐方向をFORK先とし、この条件分岐箇所を、FORK箇所を選定する。その理由は、ここで選定した分岐方向を新スレッドとしてFORKする方がもう一方をFORKする場合に比べ、FORK直後に、データ依存で停止してしまう可能性が低い、からである。

【0071】ステップ32で、データ依存の距離を求める際、単純な命令ステップ数ではなく、各命令が目的アーキテクチャのプロセッサ上で実行される際に要するものと推定されるサイクル数（クロックサイクル等）を用いることもできる。これにより、データ依存が発生する時期の遅い方を選択する、という、上記ステップ33の選択がよりの確なものになる。

【0072】ステップ34では、その直前のステップ33で、データ依存の距離の最小値に一定以上の差がない場合、元の中間プログラム内で分岐先であった側（分岐命令のtaken側（分岐成立側））をFORK先とし、この条件分岐箇所を、FORK候補に選定する。このFORK先の選定は、従来の技術として挙げた上記文献等々に示されているものと同じ理由による。

【0073】以上のようにして、FORK箇所決定部21は、FORK箇所と、FORK先を決定する。

【0074】本発明の一実施の形態では、図4に詳細を示した、FORK箇所決定部21の処理によって、通常は、最適化の最終段で行うレジスタ割り当て処理を待たずに、レジスタ割り当て情報を参照して中間プログラム上で並列化処理を行うことが可能となる。

【0075】また、本発明の一実施の形態においては、図4のステップ32、ステップ33の処理を設けることで、ループ構造以外の部分での性能向上の可能性が高まる。

【0076】図5は、本発明の一実施の形態の並列化装置11の命令並べ換え部23の動作を説明するための流れ図である。また、図6は、命令並べ換え部23の命令並べ換え処理の説明を補足する図である。以下、図5及び図6を参照しながら、命令並べ換え部23の動作を説明する。

【0077】図5を参照すると、命令並べ換え部23は、FORK箇所決定部21が決定したFORK箇所を

含む基本ブロック毎に、ステップ40からステップ44の一連の処理を行う。なお、ステップ40からステップ44の処理は、すべて中間プログラムに対して行う。これらのステップの説明に現れる命令は、すべて中間プログラム上の対応する命令を指す。

【0078】図6（A）は、命令並べ換え部23が処理対象とする基本ブロックの構造を模式的に示した図である。図6（A）における「M:=…」はメモリへのストアを行う命令を意味する。

10 【0079】まず、図5のステップ40にて、中間プログラム中の各中間項及び変数がレジスタに対応付けられるかメモリに対応付けられるかを調べる。これは、図4のステップ32と同様に、レジスタ割り当てを途中まで試行することで判定する。ステップ32からこのステップ40までの処理では、FORK先を決定するのみで、中間プログラム自体は変更されず、レジスタ割り当て状況に差が出ないので、ステップ32のときのレジスタ割り当て試行結果を保存しておき、その情報をステップ40で参照するのがよい。

20 【0080】次のステップ41では、現在処理対象としている基本ブロックの末尾にある分岐命令を、制御投機モードFORK命令に置換（変換）する。制御投機FORK命令のオペランド、即ちFORK先は、FORK箇所決定部21が選択したFORK先とする。図6（B）は、ステップ41までの処理を終えた段階の基本ブロック構造を示す図である。

30 【0081】ステップ42では、中間プログラム中で、制御投機FORK命令の直前にある分岐条件計算文（分岐条件式）を、制御投機FORK命令直後に移動させると共に、その移動先の直後、即ち当該基本ブロックの末尾に、「分岐条件成立時には自スレッドを終了させて子スレッドを確定モード（非制御投機モード）に移行させ、分岐条件非成立時には、子スレッドを破棄（アボート）させて自スレッドが後続命令列の実行を続ける」というような一群の命令列を挿入する。図6（C）は、ステップ42までの処理を終えた段階の基本ブロック構造を示す図である。

40 【0082】ステップ43では、現在処理対象としている基本ブロックにおいて、FORK命令よりも手前、即ち、上流側にある文のうち、ステップ40において、メモリに対応付けられた中間項及び変数への代入となる文の各々について、その代入文を、FORK命令よりも後方、即ち、下流側へ移動させると共に、FORK命令の直前に、ブロック設定命令を挿入し、代入文（:=）の移動先の直後に、ブロック解除命令を挿入する。ここで、挿入するブロック設定命令、及びブロック解除命令のオペランドは、移動させた代入文の代入先中間項／変数を表すメモリアドレスとする。

50 【0083】中間プログラムの形態によっては、この段階で、具体的なメモリアドレスが確定していない場合が

あるが、その場合は、中間プログラム中での中間項／変数表現と同様の形式を用いてブロック設定命令及びブロック解除命令を表現し、後に、中間プログラムからコード生成を行う際に実効メモリアドレスを示す命令列に変換すればよい。

【0084】ステップ43では、ステップ40にてレジスタに対応付けられた中間項及び変数への代入となる文のうち、FORK先で参照される可能性がある中間項／変数への代入となる文は、FORK命令後に移動させてはならない。その理由は、レジスタの値はFORK時点で子スレッドに継承されるため、FORK後に親スレッドが定義したレジスタ値は子スレッド側に渡されないためである。

【0085】このように、必ずしも全ての文をFORK命令後に移動できるわけではないため、ステップ43において、メモリへの代入文をFORK後に移動させる際には、移動させようとしている文と、その後続命令との間のデータ依存関係を調べ、移動によって実行順序が入れ替わっても、移動前と同じ演算結果が得られる場合に限って移動を行わなければならない。

【0086】なお、このとき必要となるデータ依存関係は、コンパイラ技術として一般的なものでよく、本発明の一実施の形態では、図2における中間プログラム入力部20が作成した制御フロー及びデータフロー解析結果を基に、調査して得ることができる。

【0087】図6(D)は、ステップ43までの処理を終えた段階の基本ブロック構造を示す図である。

【0088】図5のステップ44では、これまでのステップで各中間項及び変数のレジスタ割り当てについて仮定した情報を、現在処理対象の基本ブロックの先頭に挿入する。この情報は、実際の機械命令に対応するものではないので、中間プログラム上の疑似命令であるレジスタ割り当て指示命令を用いて記述する。

【0089】図6(E)は、上記ステップ44までの処理を終えた段階の基本ブロック構造を示す図である。

【0090】本発明の一実施の形態において、ステップ40及びステップ44を具備することを特徴の一つとしている。即ち、ステップ41のFORK変換処理に先立って、レジスタ割り当て状況を調べるステップ40と、ステップ41のFORK変換処理で仮定したレジスタ割り当て状況を、レジスタ割り当て部22に指示するステップ44と、を有しており、このため、中間項及び原始プログラム中の変数表現を用いた中間プログラムに対しても、詳細な命令並べ換えを行うことができる。

【0091】なお、ステップ41からステップ43の処理は、上記文献4（「制御並列アーキテクチャ向け自動並列化コンパイル手法」（酒井他、情報処理学会論文誌、Vol. 40、No. 5、May 1999））の第2049頁～第2050頁に記載されているものと同様のものである。

【0092】図2に戻ると、レジスタ割り当て部22は、中間プログラム中の中間項に、目的アーキテクチャが有する物理レジスタあるいはメモリ上に割り当てられた領域を対応付けるレジスタ割り当て処理を行う。

【0093】レジスタ割り当ての基本的な方法としては、中間項の使用頻度の高いものから割り当てる方法や干渉グラフの彩色（カラーリング）による割り当て方法などを用いることができる。これらの割り当て方法は、例えば上記文献5（「コンパイラ I 原理・技法・ツール」、A. V. エイホ他著、原田賢一訳、サイエンス社、1990））の第659頁～第665頁の記載が参照される。

【0094】レジスタ割り当て部22におけるレジスタ割り当て処理は、図5のステップ44で挿入したレジスタ割り当て指示命令に従う点で、通常のコンパイラでのレジスタ割り当て処理とは異なる。レジスタ割り当て方法として、例えば中間項使用頻度順のレジスタ割り当て方法を採用する場合は、レジスタ割り当て指示命令でレジスタに乘せるよう指定された各中間項の頻度を、他の中間項よりも高く設定することで、指示された中間項が物理レジスタに割り当てられる可能性を高める。

【0095】また、レジスタ割り当て方法として、カラーリングによる方法を採用する場合は、干渉グラフ中で割り当て可能な物理レジスタ総数を越える隣接節点数を持つ節点に対応する中間項の中からメモリ上に割り付ける中間項をいくつか選択する際、図5のステップ44で挿入したレジスタ割り当て指示命令でレジスタに乘せるように、指示されていない中間項を優先的に選択することで、レジスタ割り当て指示された中間項が物理レジスタに割り当てられる可能性を高める。

【0096】レジスタ割り当て部22にて、もし、レジスタ割り当て指示命令でレジスタに乘せるように指示された全ての中間項に物理レジスタを割り付けられなかった場合、それらの中間項は、メモリ領域に割り当てられるようにする。その理由は、命令並べ換え部23によって、レジスタ割り当てを前提としている中間項はFORK命令より手前（上流側）で値が確定するようにプログラムが変換されているため、それらの中間項が、たとえメモリ上に割り当てられたとしてもプログラムの実行結果は、変化しないからである。

【0097】逆に、レジスタ割り当て部22は、レジスタ割り当て指示命令でレジスタに乘せるよう指示されていない中間項をレジスタに割り当てないように、レジスタ割り当て処理を行う。これは、従来からのレジスタ割り当て処理における、アドレスを参照されている変数や、volatile変数等のレジスタ上に配置できない変数に対する処理と同様の枠組で対処できる。

【0098】

【実施例】次に、上記した本発明の一実施の形態の動作についてさらに詳細に説明すべく、具体的な実施例を用

いて説明する。

【0099】[実施例1] 図8は、本発明の一実施例として、並列化装置11によって並列化される中間プログラムの一例を示す図である。ここでt1～t28は中間項であり、I、J、K、R、Xは原始プログラム中で宣言された変数である。

【0100】「:=」は、右辺の値を左辺で示す場所に格納することを意味する。

【0101】「&」は、変数の置かれるメモリアドレスを返す前置演算子である。

【0102】「mem(a)」は、aの値をアドレスとするメモリの内容を示し、「:=」の右辺ならメモリロード、左辺ならメモリストアを意味する。

【0103】L1、L2、L3はラベルである。

【0104】図8の左端(の列)に示されている括弧付きの数字(1)～(37)は、説明の都合により付した番号であり、右端の(B1)～(B3)は基本ブロック番号を示す。

【0105】図7は、本実施例の動作の説明をする際に用いる中間プログラム上の制御並列関連命令一覧を示す図である。

【0106】・SPFORK 1 は、オペランド1から実行開始する投機(制御)モード子スレッドを生成する。

・TTERM c は、オペランドcが真の場合、自スレッドを終了し、子スレッドを確定する。

・FTERM c は、オペランドcが偽の場合、自スレッドを終了し、子スレッドを確定する。

・THABORT は、投機モードの子スレッドを破棄する。

・BLOCK m は、オペランドmで指定したメモリアドレスをブロック指定する。

・RELEASE m は、オペランドmで指定したメモリアドレスに設定したブロックを解除する。

・DSPIN は、後続のFORKで生成した子スレッドをデータ依存投機モードで生成する。

・DSPOUT は、子スレッドのデータ依存投機モードを解除する。RDCL t1, ... は、オペランドt1, ...で指定した中間項/変数をレジスタに割り当てるように指示する。MDCL t1, ... は、オペランドt1, ...で指定した中間項/変数をメモリに割り当てるように指示する。

【0107】図2を参照すると、FORK箇所決定部21が、図8に示された中間プログラムを受け取り、FORK箇所の決定を行う。

【0108】図3のステップ25にて、レジスタ割り当てが試行され、t1～t28の全ての中間項、及び変数I、K、Rには物理レジスタが割り当てられたが、変数J、Xには、メモリ上の領域が割り当てられた、という試行結果を得たとする。

【0109】図3のステップ26では、条件分岐命令を含む基本ブロック(B1)が並列化変換の対象となる。以下、図4を参照して、FORK箇所決定処理部27の動作を説明する。

【0110】ステップ31で、ループ構造チェックを行うが、図8の(11)の条件分岐は、ループ戻り枝ではないため、ステップ32に進む。

【0111】ステップ32では、図8の基本ブロック(B1)と(B2)、(B1)と(B3)の間でのデータ依存の距離を求める。

【0112】まず(B1) - (B2)間のデータ依存を見てみると、(B1)の(7)におけるメモリストアと、(B2)の(14)、(18)、(19)におけるメモリロードとの間にデータ依存がある。

【0113】これらのメモリアクセスは、実際には、配列要素X[I]へのストアと、変数J、配列要素X[J]からのロードであり、配列Xと、変数Jのメモリ領域が重ならず、IとJの値が異なっていれば、データ依存は発生しない。すなわちブロック(B1)において、(11)のt1:=&Xは、配列Xの先頭アドレスを中間項t1に格納し、1配列要素は4バイトであり(t4=t2*t3)、t5=t1+t4により、(7)のmem(t5)が、X[I]のメモリアドレスを示し、このアドレスへのt6のメモリストアであり、またブロック(B2)において、(18)のt15:=mem(t14)は、X[J]からのメモリロードを表している。

【0114】しかし、ここでは、データ依存解析の結果、このような条件が常に満たされる確証を得られず、メモリストアとメモリロードとの間に、潜在的なメモリデータ依存が存在すると判断されたものとして、説明を続ける。

【0115】(B1) - (B2)間の各データ依存の距離は、図8の中間プログラム上のステップで数えて、各々1、5、6である。ここで、(12)にあるラベルL1は、非実行文であるためステップ数に含めず、(13)にある命令を0、(14)にある命令を1、(15)にある命令を2、というように数えている。これにより、(B1) - (B2)間のデータ依存の距離の最小値は1である。

【0116】同様に、(B1) - (B3)間では、(B1)の(7)と、(B3)の(29)、(33)との間にデータ依存があり、その距離は、各々5、9である。つまり、(B1) - (B3)間のデータ依存の距離の最小値は5である。

【0117】ステップ33では、上で求めたデータ依存の距離の最小値である6と10とを比較し、その差が十分であると判断し、データ依存の距離の最小値の大きい側である(B3)側の分岐方向を、FORK先に選定する。

【0118】もし、図3のステップ25において、変数Jがメモリではなくレジスタに割り当てられるという見通しを得ていたとすると、(B1) - (B2)、(B1) - (B3)のデータ依存の距離の最小値は各々5、9となり、ステップ33にて(B2)側をFORK先に選定することになる。

【0119】次に、図5を参照して、命令並べ換え部23の動作を具体的に説明する。本実施例では、中間プログラム上で、図7に示す制御並列関連命令が利用できるものとする。

【0120】先の説明で例に挙げた中間プログラムを与えると、命令並べ換え部23は、基本ブロック(B1)から(B3)へのFORKを行うように、命令並べ換え処理を行う。

【0121】図5のステップ40では、中間項t1~t28と、変数I、K、Rがレジスタに割り当てられる、という情報を取得する。

【0122】図5のステップ41では、図8における(11)の分岐命令(then 以下のgoto L2)を、制御投機FORK命令であるSPFORK命令に置き換える。SPFORK命令のオペランドは、FORK先として決定した(B3)、即ちラベルL2とする(図9の(58)のSPFORK L2)。

【0123】図5のステップ42では、図8における、分岐条件を計算する文を構成する一連の命令(8)~(10)を、SPFORK命令の直後に移動させ、(11)の分岐条件に合わせた条件付きスレッド終了命令である、FTERM命令(条件不成立時に自スレッドを終了する命令)、更には、THABORT命令と、後続ブロックである(B1)への無条件分岐命令(goto L1)を挿入する。

【0124】これらの命令列は、以下のように動作するように挿入する。

【0125】即ち、SPFORK後に、分岐条件計算を行って、SPFORKが本当に正しかったか否かを判定する。

【0126】SPFORKが正しい(投機成功)ならば、条件付きスレッド終了命令の一種FTERMにより、自スレッド(親スレッド)を終了させると同時に、生成した子スレッドを、制御投機モードから確定モード(非制御投機モード)に変更する。

【0127】一方、SPFORKが正しくなかった(投機失敗)ならば、FTERM命令は何もせず、後続のTHABORT命令によって、子スレッドを破棄し、無条件分岐命令(図9の(64)のgoto L1)によって、自スレッドは、FORK先とは逆の分岐先の命令の実行を続けるようにする。図9は、図8に示す中間プログラムに対して、図5のステップ42までの変換を施した中間プログラムを示す図である。

【0128】図5のステップ43では、メモリストア文

を構成する一連の命令である図9の(56)と(57)を、SPFORK命令(58)の直後に移動し(図10の(108)、(109))、その直後に、RELEASE命令を挿入する(図10の(110))。

【0129】RELEASE命令のオペランドは、移動させたストア命令(57)のオペランドと同じ中間項を用いる。またSPFORK命令(58)の直前に、ストア命令(57)のオペランドと同じ中間項をオペランドとするBLOCK命令を挿入する。図10は、ここまでの変換を施した中間プログラムである。

【0130】ここで、メモリストア文をFORK後に移動させる際、命令(51)~(57)全部ではなく、(56)と(57)のみを移動させた理由は、実効メモリアドレスを保持する中間項を、BLOCK命令及びRELEASE命令で再利用できるようにするためである。コンパイラ分野において、「共通部分式削除」と呼ばれる最適化機能を用いれば、ここでBLOCK及びRELEASEのたびに実効アドレスを再計算するようなプログラム変換を施しても、共通部分式削除最適化により冗長な実効アドレス計算は削除される。

【0131】図5のステップ44では、各基本ブロック(B1)、(B2)、(B3)の先頭に、RDCL命令及びMDCL命令を挿入する。これらの命令は、後段のレジスタ割り当て部22への指示であり、このうち、RDCL命令は、指定した変数ないし中間項へのレジスタ割り当てを行い、MDCL命令は、指定した変数ないし中間項へのメモリ領域の割り当てを行う。

【0132】図11は、ここまでの変換を施した中間プログラムである。(201)~(203)、(221)~(223)、(235)~(237)がステップ44で挿入した命令である。

【0133】再度、図2を参照すると、命令並べ換え後、レジスタ割り当て部22が、レジスタに置くべき変数や中間項に物理レジスタを割り当てる。

【0134】本実施例では、プログラム変換装置から出力される目的プログラムが実行される装置のアーキテクチャ(ターゲットアーキテクチャ)は、r0~r31の32本の物理レジスタを持ち、レジスタ変数(原始プログラム中の変数のうち、物理レジスタに割り当てられるもの)用に、物理レジスタr10~r19、作業用レジスタ(主として演算途中結果を保持するレジスタ)用に、物理レジスタr20~r30を用いるようなレジスタ割り当てを行うものとする。

【0135】レジスタ割り当て部22は、
・変数I、K、Rに、それぞれr11、r13、r12を割り当て、

・中間項t1~t28には、r20~r30を順に割り当てる。

【0136】図12は、図11に示した中間プログラムに対して、レジスタを割り当てた後の中間プログラムを

示している。なお、図 12 の中間プログラムには、冗長な命令が含まれるが、これらの冗長さは、従来から提案されている最適化処理を行うことで除去できる。従来の各種最適化方法については、例えば文献 5 (「コンパイラ I I 原理・技法・ツール」、A. V. エイホ他著、原田賢一訳、サイエンス社、1990) の第 772 頁〜第 790 頁に説明されている。

【0137】これらの最適化処理は、レジスタ割り当て前に施すことが好ましいため、図 2 において、好ましくは、FORK 箇所決定部 21 よりも手前、例えば中間プログラム入力部 20 の内部に実装される。

【0138】【実施の形態 2】次に本発明の第 2 の実施の形態について、図面を参照して詳細に説明する。本発明の第 2 の実施の形態では、プロファイル情報を参照して、FORK 先及び FORK 時のデータ依存対処方法を選択する点に特徴があるが、その他の点は、前記第 1 の実施の形態と、基本的に同一とされる。

【0139】図 1 を参照すると、本発明の第 2 の実施の形態においては、まずプログラム変換装置 2 を用いて原始プログラム 1 から目的プログラム 3 を生成する。このとき、プログラム変換装置 2 は、機械命令アドレスとプログラム変換装置 2 内で用いる中間プログラムとの対応をとるための情報を、目的プログラム 3 の中に埋め込む。目的プログラム実行装置 4 は、この目的プログラム 3 を実行し、その際に収集した実行情報をプロファイル情報ファイル 5 として出力する。

【0140】次に、再度プログラム変換装置 2 を用いて、原始プログラム 1 を並列化し、目的プログラム 3 を生成する。このとき、プロファイル情報ファイル 5 を用いて、より実行時性能の高い目的プログラム 3 を生成する。

【0141】プログラム変換装置 2 の構成は、前記第 1 の実施の形態とほぼ同様であるが、図 2 における、FORK 箇所決定部 21 と、命令並べ換え部 23 の動作が、前記第 1 の実施の形態とは異なり、プロファイル情報を参照しながら、より実行時性能の期待できる並列化処理を行う。

【0142】本発明の第 2 の実施の形態における FORK 箇所決定部 21 の動作概要は、図 3 を参照して説明した前記第 1 の実施の形態と同じである。即ち、FORK 箇所決定部 21 は、与えられた中間プログラム中の各関数毎に、図 3 のステップ 26 で当該関数内でレジスタ割り当てを試行した後、ステップ 27 で、当該関数内に含まれる各条件分岐命令に対して、FORK 箇所決定処理を行う。

【0143】図 13 は、各条件分岐命令に対する FORK 箇所決定処理 (図 3 のステップ 26) の第 2 の実施の形態を示す流れ図である。

【0144】ステップ 50 では、当該条件分岐命令が入力中間プログラム中のループ構造の戻り枝に相当する分

岐命令であるか否かを判定する。この処理は、前記第 1 の実施の形態のステップ 31 (図 4 参照) と同じものである。

【0145】もし、ループ構造の戻り枝なら、ステップ 56 で、その戻り枝方向を FORK 先として仮決定する。これは FORK するとすれば、こちらの方向を、FORK 先とするのがよい、という意味であり、実際に FORK することにするか否かは後段で決定する。

【0146】ステップ 51 では、入力したプロファイル情報に基づいて、当該条件分岐命令の taken (分岐成立) 側 / fall through 側が選択される確率を計算する。

【0147】ステップ 52 では、ステップ 51 で計算した、2 つの確率間に一定以上の違いがあるかどうかを判定する。

【0148】分岐確率の違いが、その判断基準を越えていれば、ステップ 57 にて、確率の高い側を、FORK 先として仮決定する。

【0149】ステップ 53 では、条件分岐の 2 つの分岐先各々に対して、データ依存の距離の最小値を計算する。この処理は、前記第 1 の実施の形態におけるステップ 32 (図 4 参照) と同じものである。

【0150】ステップ 54 では、ステップ 53 で、条件分岐の両側に関して求めた 2 つのデータ依存の距離の最小値を比較し、両者に一定以上の差があるか否かを判定する。

【0151】一定以上の差があるか、又はデータ依存がない場合、ステップ 55 にて、データ依存の距離の最小値の大きい側の分岐方向を FORK 先として決定する。これは、前記第 1 の実施の形態のステップ 33 (図 4 参照) と同様の処理である。

【0152】ステップ 58 及びステップ 59 では、ステップ 56 ないしステップ 57 で仮決定した FORK 先に対して、ステップ 53 と同様に、データ依存の距離の最小値を計算し、仮決定した FORK 先側のデータ依存の距離の最小値が一定以上あるか否かを判定する。

【0153】仮決定した FORK 先側のデータ依存の距離の最小値が一定以上あるか、メモリを介したデータ依存がなければ、ステップ 56 ないしステップ 57 で仮決定した FORK 先を、正式な FORK 先として確定する (ステップ 60)。

【0154】ステップ 54 ないしステップ 59 で、データ依存の距離の最小値が一定水準に満たないと判断された場合には、ステップ 67 にて、当該基本ブロックを、FORK 箇所から除外する。これは例えば、当該条件分岐命令に、FORK 対象外マークを付すなどして、後段の処理で並列化対象から外れるようにする。

【0155】ステップ 50 からステップ 60 で、当該条件分岐命令が FORK 箇所として決定され、また分岐命令のいずれかの分岐先が FORK 先も決定されたあと、

ステップ 61 にて、データ依存発生頻度を計算する。即ち、プロファイル情報に基づき、FORK 元の基本ブロック（現在処理対象としている分岐命令を末尾に含む基本ブロック）中で定義された値が FORK 先として決定した基本ブロックにて参照される回数の、このパス（現在の基本ブロックから FORK 先と決定した基本ブロックへの制御の流れ）を通過した回数に対する比率を計算する。

【0156】このデータ依存発生頻度計算においては、図 4 のステップ 32 と同様に、最終的にレジスタに割り当てられる中間項及び変数によるデータ依存は除外し、メモリ上に配置された中間項及び変数のみを計算対象とする。

【0157】ステップ 62 にて、そのデータ依存発生頻度が一定水準より高いか否かを判断し、もし高ければステップ 65 へ、低ければステップ 63 へ進む。

【0158】ステップ 63 では、中間プログラム中で、FORK 元基本ブロックから、FORK 先基本ブロックへのデータ依存を引き起こし得るメモリ上の中間項／変数の個数を数え挙げ、それが一定水準より多いか否かを判定する。この数え挙げは、プロファイル情報によらず、中間プログラム中の命令列を検査して静的に行う。その結果、データ依存箇所数が一定水準より多ければ、DSP 方式による FORK（ステップ 64）、そうでなければ BLOCK 方式による FORK（ステップ 66）を用いることとし、その情報を、中間プログラム中の FORK 命令に付与する。

【0159】ステップ 65 では、ステップ 63 と同様にデータ依存しているメモリ上の変数の個数を数え挙げる。その数が、一定水準より少なければ、BLOCK 方式による FORK（ステップ 66）を用いるが、一定水準より多ければ、当該基本ブロックは FORK 候補から外す（ステップ 67）。

【0160】図 14 は、本発明の第 2 の実施の形態における、命令並べ換え部 23 の動作を示す図である。命令並べ換え部 23 は、FORK 箇所決定部 21 が決定した FORK 箇所及び FORK 時データ保証方法に基づき、ステップ 70 からステップ 76 の一連の処理を行う。なお、ステップ 70 からステップ 76 の各処理はすべて中間プログラムに対して行う。これらのステップの説明に現れる命令は、すべて中間プログラム上の対応する命令を指す。

【0161】ステップ 70、ステップ 71、ステップ 72 は、前記第 1 の実施の形態で参照した図 5 のステップ 40、ステップ 41、ステップ 42 とそれぞれ同様の処理である。

【0162】ステップ 73 では、前段の FORK 箇所決定部 21 が決定した当該 FORK 箇所の FORK 時データ保証方式が BLOCK 方式か DSP 方式かをチェックし、前者ならステップ 74、後者ならステップ 75 に進

む。

【0163】ステップ 74 は、前記第 1 の実施の形態で参照した図 5 のステップ 43 と同様の処理である。即ち、FORK 前のメモリストア文を FORK 後に移動すると共に、必要なブロック設定及びブロック解除命令を挿入する。

【0164】移動の際には、データ依存関係を検査し、命令実行順序が入れ替わっても演算結果が不変となるもののみ移動させる、という点も、図 5 のステップ 43 の処理と同様である。

【0165】ステップ 75 もまた、メモリに対応付けられた中間項への代入文を、FORK 命令後に移動させる、という点で、図 5 のステップ 43 と類似する処理であるが、図 5 のステップ 43 では、アクセス対象のメモリアドレスをオペランドとするブロック設定命令及びブロック解除命令を挿入したのに対し、ステップ 75 では、データ依存投機モードで FORK を行うように、ステップ 71 で作成した FORK 命令を修正する点が異なる。

【0166】ステップ 76 は、前記第 1 の実施の形態で参照した、図 5 のステップ 44 と同様の処理である。

【0167】このように、本発明の第 2 の実施の形態においては、プロファイル情報を用いて、FORK するか否か、及び、FORK 時のデータ依存保証方法を決定する。このため、条件分岐において制御の流れが片方に偏っている場合には、その方向を FORK することで、制御投機 FORK の成功確率が高まる。

【0168】また、実際に発生したデータ依存の頻度や、依存箇所の数を考慮した並列化を行うことにより、並列化プログラムの実行時のオーバーヘッドを低減させ、より並列化による性能を引き出しやすくしている。

【0169】〔実施例 2〕次に、具体的な実施例を用いて、本発明の第 2 の実施の形態の動作を説明する。図 15 は、本発明の並列化装置によって並列化される中間プログラムの一例を示す図である。図 15 における記号等の意味は、図 8 に示した第 1 の実施例の中で用いられているものと同じである。また、中間プログラム上で用いる制御並列関連命令も、図 7 に示した前記第 1 の実施例のものと同じである。

【0170】図 1 を参照すると、本発明の第 2 の実施例では、プログラム変換装置 2 は原始プログラム 1 を並列化せずに目的プログラム 3 に変換する。その際、プログラム変換装置 2 は、各基本ブロックの先頭に当該基本ブロックの識別番号に基づく名前をもつラベルを挿入し、そのラベルに関するシンボル情報も目的プログラム 3 内に含めて出力する。目的プログラムにシンボル情報を埋め込むことは、コンパイラ分野で広く行われており、目的プログラムからシンボル名及びそのシンボルに対応付けられたアドレスを引き出すことができれば、任意の手法を用いることができる。

【0171】目的プログラム実行装置4は、目的プログラム3を読み込み、埋め込まれたシンボル情報を元に目的プログラム内の基本ブロックの集合を認識する。これにより、プログラム変換装置2の内部の中間プログラムを構成していた基本ブロック集合と区切り方の等価な基本ブロック集合を、目的プログラム実行装置4も認識することができる。

【0172】目的プログラム実行装置4は、読み込んだ目的プログラム3を構成する機械命令をソフトウェアで解釈して実行しながら、目的プログラムの振るまい、具体的には、(1)各条件分岐命令の条件が成立した回数と成立しなかった回数、及び、(2)制御フロー上隣接する基本ブロック間でのメモリデータ依存の回数と当該機械命令アドレス、の情報を収集する。

【0173】目的プログラム3の実行が完了した後、目的プログラム実行装置4は収集した前記情報の中の機械命令アドレスを基本ブロックの識別番号に変換し、

(1)各基本ブロックから制御フロー上でそれに続く各基本ブロックへ制御が流れた回数、及び、(2)制御フロー上隣接する基本ブロック間でメモリデータ依存を引き起こした回数、の内容を含むプロファイル情報ファイル5を出力する。

【0174】図16は、図15に示した中間プログラムに対応する目的プログラムを、目的プログラム実行装置4に与えたとき、目的プログラム実行装置4が出力したプロファイル情報ファイル5に含まれるプロファイル情報の一部を示した図である。

【0175】図16(A)は、基本ブロック間の分岐回数であり、例えば基本ブロック(B11)から、基本ブロック(B12)及び(B13)への分岐は、各々20回、180回であったことを示している。

【0176】図16(B)は、基本ブロック間メモリデータ依存回数を表し、例えば基本ブロック(B15)でストアした値を基本ブロック(B16)でロードした回数がのべ120回であったことを示している。

【0177】次に、プロファイル情報ファイル5をプログラム変換装置2に与えて、原始プログラム1を並列化する動作について説明する。

【0178】図2を参照すると、並列化装置11は、一度目に原始プログラムを変換したときと全く同じ中間プログラム6が与えられる。

【0179】中間プログラム入力部20は、中間プログラム6を読み込み、フロー解析を行った上で、FORK箇所決定部21に渡す。

【0180】他方、プロファイル情報入力部25は、目的プログラム実行装置4が生成したプロファイル情報ファイル5を読み込む。図15の中間プログラム例を用いた際にプロファイル情報入力部25が読み込んだプロファイル情報の中身は、図16に示したものである。

【0181】図2を参照すると、FORK箇所決定部2

1が、図15に示された中間プログラムを受け取り、FORK箇所の決定を行う。

【0182】図3のステップ25にて、レジスタ割り当てが試行され、t1~t43の全中間項、及び、変数J、Pには、物理レジスタが割り当てられ、変数K、X、Y、Zには、メモリ上の領域が割り当てられたとする。

【0183】図3のステップ26では、条件分岐を含む基本ブロック(B11)、(B13)、(B15)が並列化変換の対象となる。

【0184】以下、図13を参照しながら、基本ブロック(B11)、(B13)、(B15)に対するFORK箇所決定処理部27の動作を説明する。

【0185】基本ブロック(B11)は、ステップ50でループ戻り枝ではないと判定される。

【0186】ステップ51にて、図16(A)に示したプロファイル情報から、分岐確率を計算し、(B12)への分岐が10%、(B13)への分岐が90%と求められる。

【0187】ステップ52では、比率で2倍以上の偏りがあるかどうかを判断基準にとり、(B11)からの分岐確率の偏りは十分であると判定し、ステップ57により(B13)がFORK先として仮決定される。

【0188】ステップ58では、メモリデータ依存を調べるが、(B11)にはメモリストアがないため、ステップ60にて(B13)がFORK先に決定される。

【0189】次にステップ61で、データ依存発生頻度を求めるが、(B11)に関するメモリデータ依存がないため、結局、ステップ66にて、BLOCK方式が候補となる。

【0190】基本ブロック(B13)は、分岐確率が(B14)へ15%、(B15)へ85%である以外は、上の(B11)と同様のステップを辿り、結局、ステップ60にて、(B15)がFORK先と決定され、ステップ66にて、BLOCK方式が候補となる。

【0191】基本ブロック(B15)の場合、分岐確率が(B16)へ15%、(B17)へ85%であり、ステップ52では、分岐確率の偏りが十分であると判定され、ステップ57で(B17)がFORK先に仮決定される。

【0192】(B15)から(B17)へのデータ依存の距離の最小値は6であり、ステップ59における判定基準をデータ依存の距離4以上とすると、ステップ60にて(B17)がFORK先として決定する。

【0193】ステップ61にて、図16(A)及び図16(B)に示したプロファイル情報を基にして、(B15)から(B17)へのデータ依存発生頻度を求めると、4/170で約2.4%となる。

【0194】ステップ62での発生頻度の判定基準を30%とすると、この発生頻度は、低いと判断される。

【0195】ステップ63では、データ依存を起こしうるメモリアクセス箇所を中間プログラム中から数え挙げる。

【0196】(B15)には、memが、左辺中に2箇所あるほか、メモリ上に割り当てられる変数Kが、左辺に1箇所現れる。

【0197】これらのメモリストアは、すべて(B17)でのメモリロードとアドレスが重なる可能性があり、計3箇所のメモリ依存箇所があることになる。

【0198】ステップ63での依存メモリ箇所数の判定基準を3以上とすると、(B15)の場合は依存箇所が多いと判定され、ステップ64にて、FORK方式としてDSP方式が候補となる。

【0199】ここで、もし、(B15)から(B16)及び(B17)への分岐確率が各々40%、60%であったと仮定すると、ステップ52では、分岐確率の偏りが小さいと判定される。

【0200】ステップ53で、データ依存の距離を求めると、(B15)から(B16)へのデータ依存の距離の最小値は5、(B15)から(B17)へのデータ依存の距離の最小値も5となり、ステップ54では、依存距離の差が小さいと判定される。

【0201】そのため、ステップ67にて、(B15)は、FORK箇所の候補から除外されることになる。

【0202】次に、図14を参照して、第2の実施例における命令並べ換え部23の動作を具体的に説明する。

【0203】命令並べ換え部23は、基本ブロック(B11)、(B13)、(B15)の各条件分岐に対し、図14に示された一連の処理を行う。

【0204】基本ブロック(B11)については、ステップ71で、(B13)へのSPFORK命令が作られ、ステップ72で、条件式計算のための命令群がSPFORK後に移される。

【0205】(B11)からのFORKでのデータ依存保証は、BLOCK方式を候補としているが、ブロックすべきメモリアクセスが存在しないため、ステップ74でBLOCK命令やRELEASE命令を挿入することはない。

【0206】ステップ76では、レジスタに乗る予定の中間項t1、t2、t3をオペランドとしたRDCL命令を挿入する。

【0207】基本ブロック(B13)も、BLOCK方式のデータ依存保証を行うため、前述の基本ブロック(B11)と同様の流れで処理される。

【0208】基本ブロック(B15)の場合、データ依存保証をDSP方式で行うため、図14のステップ75で、メモリストア命令群の移動が行われ、データ依存投機モードを指示する命令が挿入される。

【0209】具体的には、mem(t12)、mem(t17)、及び、変数Kへのストアを構成する文が、

SPFORK命令後に移され、その直後に、データ依存投機モードの終了を指示するDSPOUTが挿入される。

【0210】またSPFORK命令直前には、データ依存投機モードで子スレッドを生成することを指示するDSPIN命令が挿入される。

【0211】図17は、本発明の第2の実施例における、命令並べ換え部23の処理を終えた後の中間プログラムを示す図である。

【0212】なお、これまで説明した各実施の形態は、他のFORK箇所/FORK先決定方法と組み合わせて実施することも可能である。例えば文献4(「制御並列アーキテクチャ向け自動並列化コンパイル手法」(酒井他、情報処理学会論文誌、Vol. 40, No. 5, May 1999))の第2049頁～第2050頁では、FORK命令が元の条件分岐命令よりも何命令上流に移動できるかというFORKブースト値をFORK箇所選定に利用する方法が開示されている。この方法を導入するには、前記第1の実施の形態及び前記第2の実施の形態における、命令並べ換え部23の中のステップ44ないしステップ76の直前に、FORKブースト値によってFORKを行うか否かの判定処理を組み込めばよい。

【0213】前記第1及び第2の実施の形態における、プログラム変換装置(コンパイラ)2の並列化装置11におけるFORK箇所決定部21、レジスタ割り当て部22、命令並べ換え部23、中間プログラム出力部24、プロファイル情報入力部25は、コンピュータ上で実行されるプログラムにより、その機能・処理が実現される。この場合、該プログラムを記録した記録媒体(CD-ROM、DVD(digital versatile disk)、FD(フロッピー(登録商標))、HDD(ハードディスク)、MT(磁気テープ)、半導体メモリ)から、該プログラム(実行形式)をコンピュータの主記憶にロードして実行するか、あるいは、サーバ等から通信媒体を介して、コンピュータのHDD等にダウンロードしてインストールし、該プログラムを実行することで、本発明のプログラム変換装置を実施することができる。

【0214】

【発明の効果】以上説明したように、本発明によれば下記記載の効果を奏する。

【0215】本発明の第1の効果は、中間プログラムレベルでFORK命令を用いた並列化を的確に行える、ということである。

【0216】その理由は、本発明においては、レジスタ割り当て部は並列化よりも後段に位置するにも関わらず、並列化処理にてレジスタ割り当てを試行し、各中間項がレジスタに乗るのかメモリ領域に格納されるのかの予測ができるからである。

【0217】本発明の第2の効果は、FORK命令を用いて並列実行した場合の性能が向上する、ということ

ある。

【0218】その理由は2つある。一つは、本発明においては、FORK箇所決定部が静的に親子スレッド間でのデータ依存の状況を調べ、データ依存による子スレッド実行の一時停止の可能性が低くなるようにFORK先を選定するからである。もう一つは、FORK箇所決定部が、プロファイル情報に基づいて、動的なデータ依存発生状況を調べ、データ依存による子スレッド実行の一時停止や再実行の可能性が低くなるようにFORK先を選定するからである。

【図面の簡単な説明】

【図1】本発明の実施の形態の全体構成を示す図である。

【図2】本発明の実施の形態におけるプログラム変換装置の内部構成を示す図である。

【図3】本発明の第1の実施の形態におけるFORK箇所決定部の動作を示す図である。

【図4】本発明の第1の実施の形態におけるFORK箇所決定処理部の動作を示す図である。

【図5】本発明の第1の実施の形態における命令並べ換え部の動作を示す図である。

【図6】本発明の第1の実施の形態において命令が並べ換えられる様子を示す図である。

【図7】本発明の第1及び第2の実施例における中間プログラム上の制御並列関連命令一覧を示す図である。

【図8】本発明の第1の実施例における並列化前の中間プログラムを示す図である。

【図9】本発明の第1の実施例において命令並べ換え途中の中間プログラムを示す図である。

【図10】本発明の第1の実施例において命令並べ換え途中の中間プログラムを示す図である。

【図11】本発明の第1の実施例において命令並べ換えを終えた中間プログラムを示す図である。

【図12】本発明の第1の実施例においてレジスタ割り当てを終えた中間プログラムを示す図である。

【図13】本発明の第2の実施の形態におけるFORK箇所決定処理部の動作を示す図である。

【図14】本発明の第2の実施の形態における命令並べ換え部の動作を示す図である。

【図15】本発明の第2の実施例における並列化前の中間プログラムを示す図である。

【図16】本発明の第2の実施例におけるプロファイル情報を示す図である。

【図17】本発明の第2の実施例において命令並べ換えを終えた中間プログラムを示す図である。

【図18】周知のMUSCATアーキテクチャのFORK命令を説明するための図である。

【図19】周知のMUSCATアーキテクチャのBLOCK方式説明のための図である。

【図20】周知のMUSCATアーキテクチャの制御投機モード説明のための図である。

【図21】周知のMUSCATアーキテクチャのデータ依存投機モード説明のための図である。

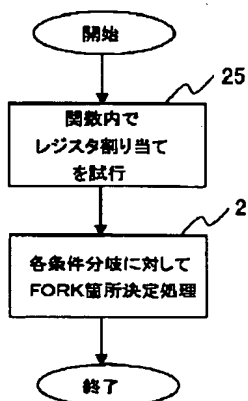
【符号の説明】

- 1 原始プログラム
- 2 プログラム変換装置
- 3 目的プログラム
- 4 目的プログラム実行装置
- 5 プロファイル情報ファイル
- 6 中間プログラム
- 7 中間プログラム
- 10 構文解析装置
- 11 並列化装置
- 12 コード生成装置
- 20 中間プログラム入力部
- 21 FORK箇所決定部
- 22 レジスタ割り当て部
- 23 命令並べ換え部
- 24 中間プログラム出力部
- 25 プロファイル情報入力部

【図3】

【図7】

【図16】



FFORK 1	1から実行開始する投機モード子スレッドを生成
FFORK 0	0が真なら自スレッド終了、子スレッド確定
FFORK 0	0が偽なら自スレッド終了、子スレッド確定
FFORK 0	投機モードの子スレッドを生成
BLOCK =	■で指定したメモリアドレスをブロック指定
RELEASE =	■で指定したメモリアドレスに設定したブロックを解除
DEPIN	後続のFORKで生成した子スレッドをデータ依存投機モードで生成
DEPOUT	子スレッドのデータ依存投機モードを解除
ADCL t, ...	t, ...で指定した中間項変数をレジスタに割り当てるよう指示
MDCL t, ...	t, ...で指定した中間項変数をメモリに割り当てるよう指示

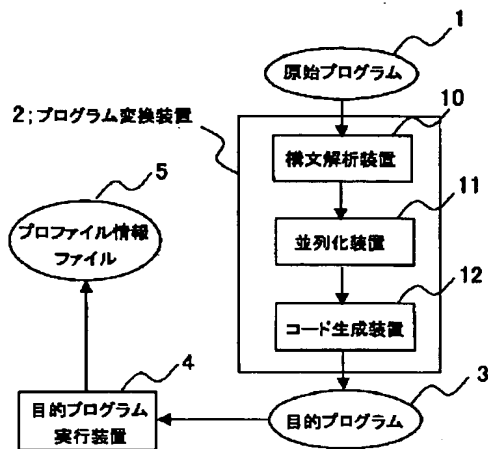
分岐変数		
B11	B12: 20	B13: 180
B13	B14: 30	B15: 170
B15	B16: 30	B17: 170

(A)

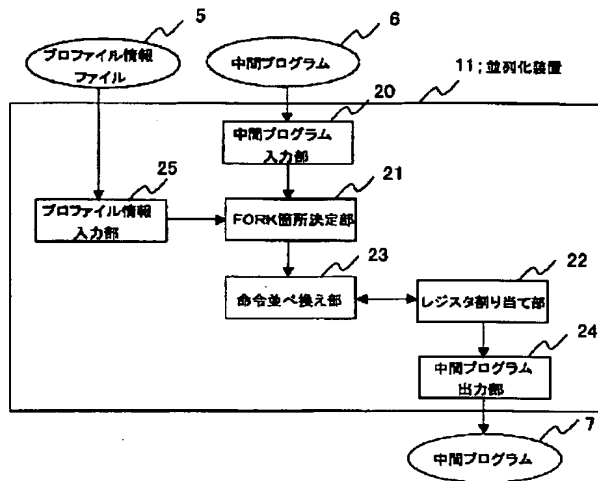
メモリデータ依存	
B15 → B16	120
B15 → B17	4

(B)

【図1】



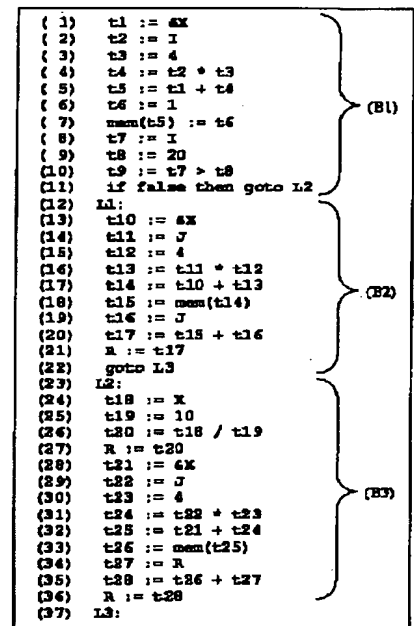
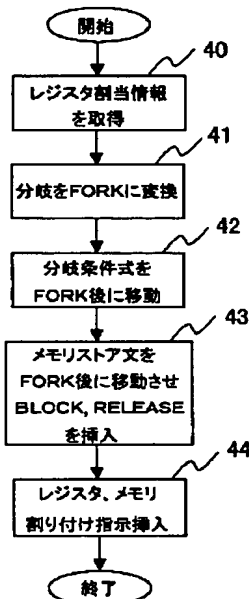
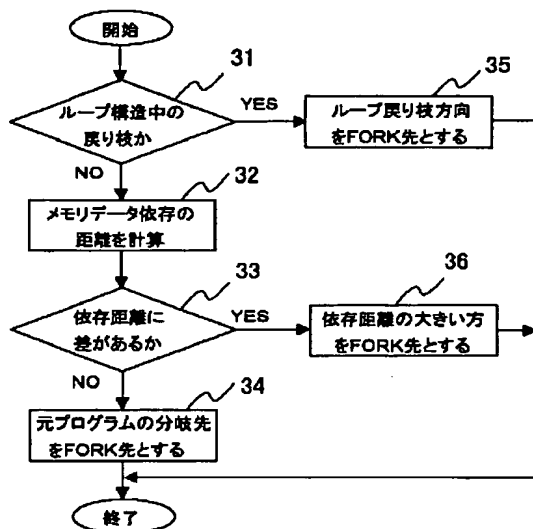
【図2】



【図5】

【図8】

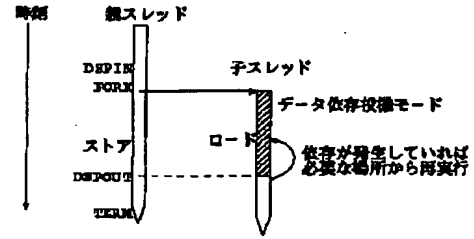
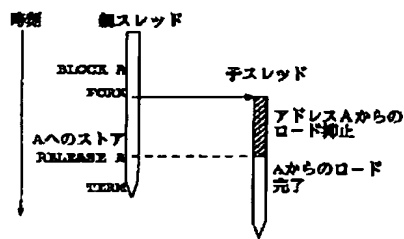
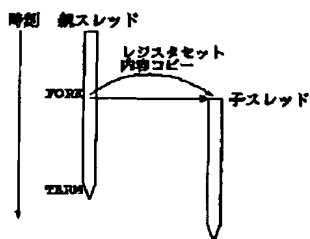
【図4】



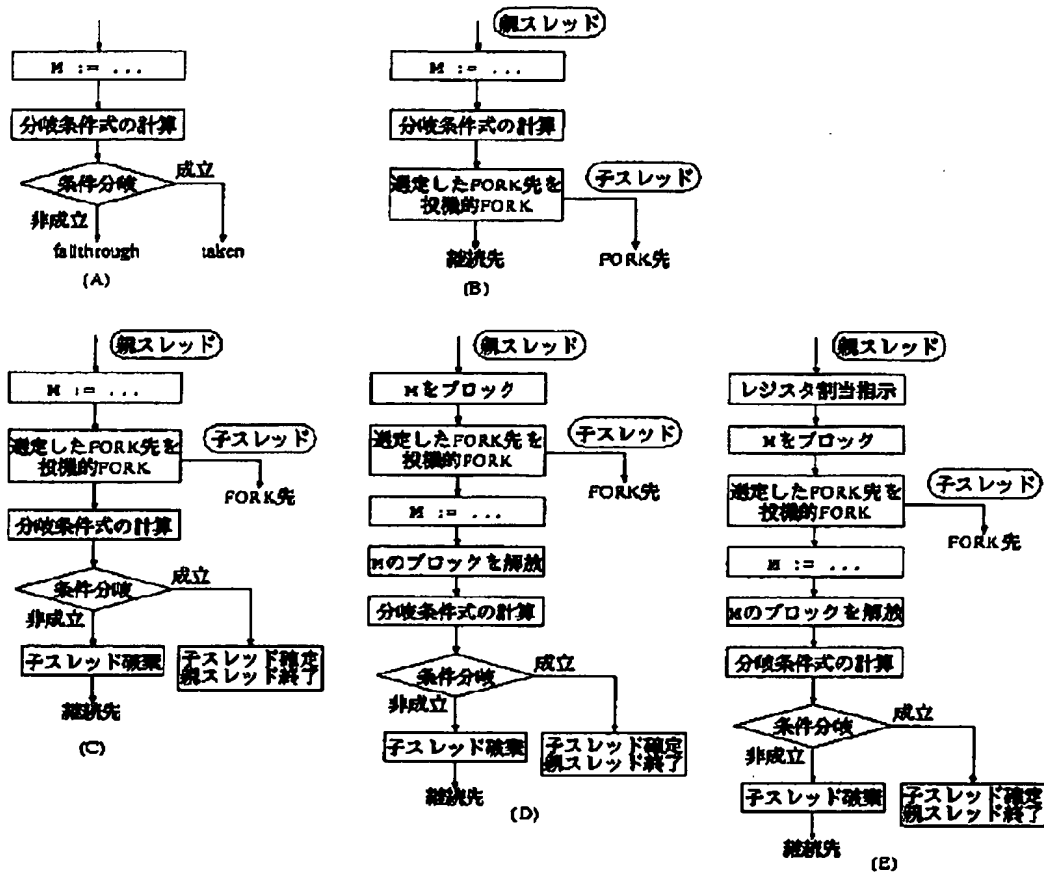
【図18】

【図19】

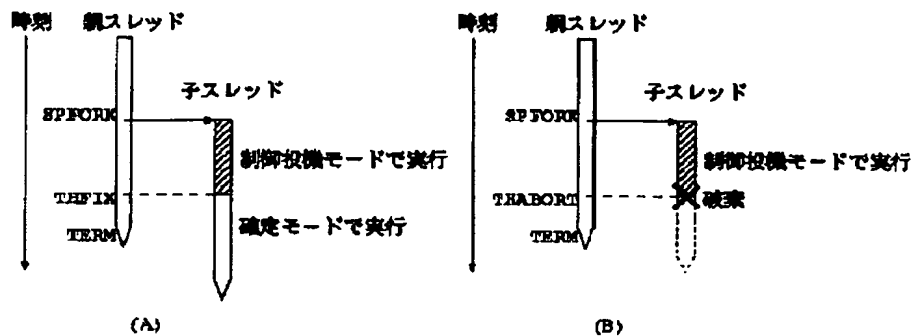
【図21】



【図 6】



【図 20】



【図9】

```

(51) t1 := AX
(52) t2 := 1
(53) t3 := 4
(54) t4 := t2 * t3
(55) t5 := t1 + t4
(56) t6 := 1
(57) mem(t5) := t6
(58) SPFORK L2
(59) t7 := 1
(60) t8 := 20
(61) t9 := t7 > t8
(62) PTERM
(63) THABORT
(64) goto L1
(65) L1:
(66) t10 := AX
(67) t11 := J
(68) t12 := 4
(69) t13 := t11 * t12
(70) t14 := t10 + t13
(71) t15 := mem(t14)
(72) t16 := J
(73) t17 := t15 + t16
(74) R := t17
(75) goto L3
(76) L2:
(77) t18 := K
(78) t19 := 10
(79) t20 := t18 / t19
(80) R := t20
(81) t21 := AX
(82) t22 := J
(83) t23 := 4
(84) t24 := t22 * t23
(85) t25 := t21 + t24
(86) t26 := mem(t25)
(87) t27 := R
(88) t28 := t26 + t27
(89) R := t28
(90) L3:

```

【図10】

```

(101) t1 := AX
(102) t2 := 1
(103) t3 := 4
(104) t4 := t2 * t3
(105) t5 := t1 + t4
(106) BLOCK t5
(107) SPFORK L2
(108) t6 := 1
(109) mem(t5) := t6
(110) RELEASE t5
(111) t7 := 1
(112) t8 := 20
(113) t9 := t7 > t8
(114) PTERM
(115) THABORT
(116) goto L1
(117) L1:
(118) t10 := AX
(119) t11 := J
(120) t12 := 4
(121) t13 := t11 * t12
(122) t14 := t10 + t13
(123) t15 := mem(t14)
(124) t16 := J
(125) t17 := t15 + t16
(126) R := t17
(127) goto L3
(128) L2:
(129) t18 := K
(130) t19 := 10
(131) t20 := t18 / t19
(132) R := t20
(133) t21 := AX
(134) t22 := J
(135) t23 := 4
(136) t24 := t22 * t23
(137) t25 := t21 + t24
(138) t26 := mem(t25)
(139) t27 := R
(140) t28 := t26 + t27
(141) R := t28
(142) L3:

```

【図11】

```

(201) BLOCK t1-t9
(202) BLOCK t1
(203) BLOCK t1
(204) t1 := AX
(205) t2 := 1
(206) t3 := 4
(207) t4 := t2 * t3
(208) t5 := t1 + t4
(209) BLOCK t5
(210) SPFORK L2
(211) t6 := 1
(212) mem(t5) := t6
(213) RELEASE t5
(214) t7 := 1
(215) t8 := 20
(216) t9 := t7 > t8
(217) PTERM
(218) THABORT
(219) goto L1
(220) L1:
(221) BLOCK t10-t17
(222) BLOCK t1
(223) BLOCK t1, J
(224) t10 := AX
(225) t11 := J
(226) t12 := 4
(227) t13 := t11 * t12
(228) t14 := t10 + t13
(229) t15 := mem(t14)
(230) t16 := J
(231) t17 := t15 + t16
(232) R := t17
(233) goto L3
(234) L2:
(235) BLOCK t18-t28
(236) BLOCK t1
(237) BLOCK t1, J
(238) t18 := K
(239) t19 := 10
(240) t20 := t18 / t19
(241) R := t20
(242) t21 := AX
(243) t22 := J
(244) t23 := 4
(245) t24 := t22 * t23
(246) t25 := t21 + t24
(247) t26 := mem(t25)
(248) t27 := R
(249) t28 := t26 + t27
(250) R := t28
(251) L3:

```

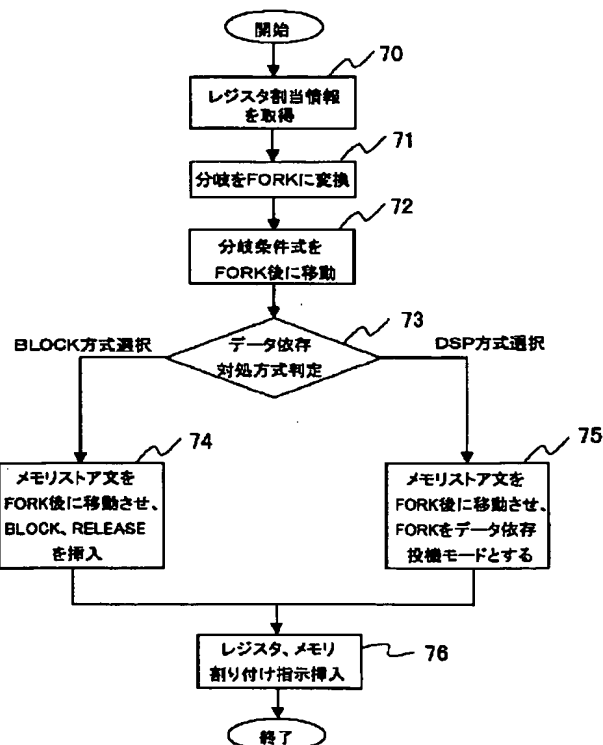
【図12】

```

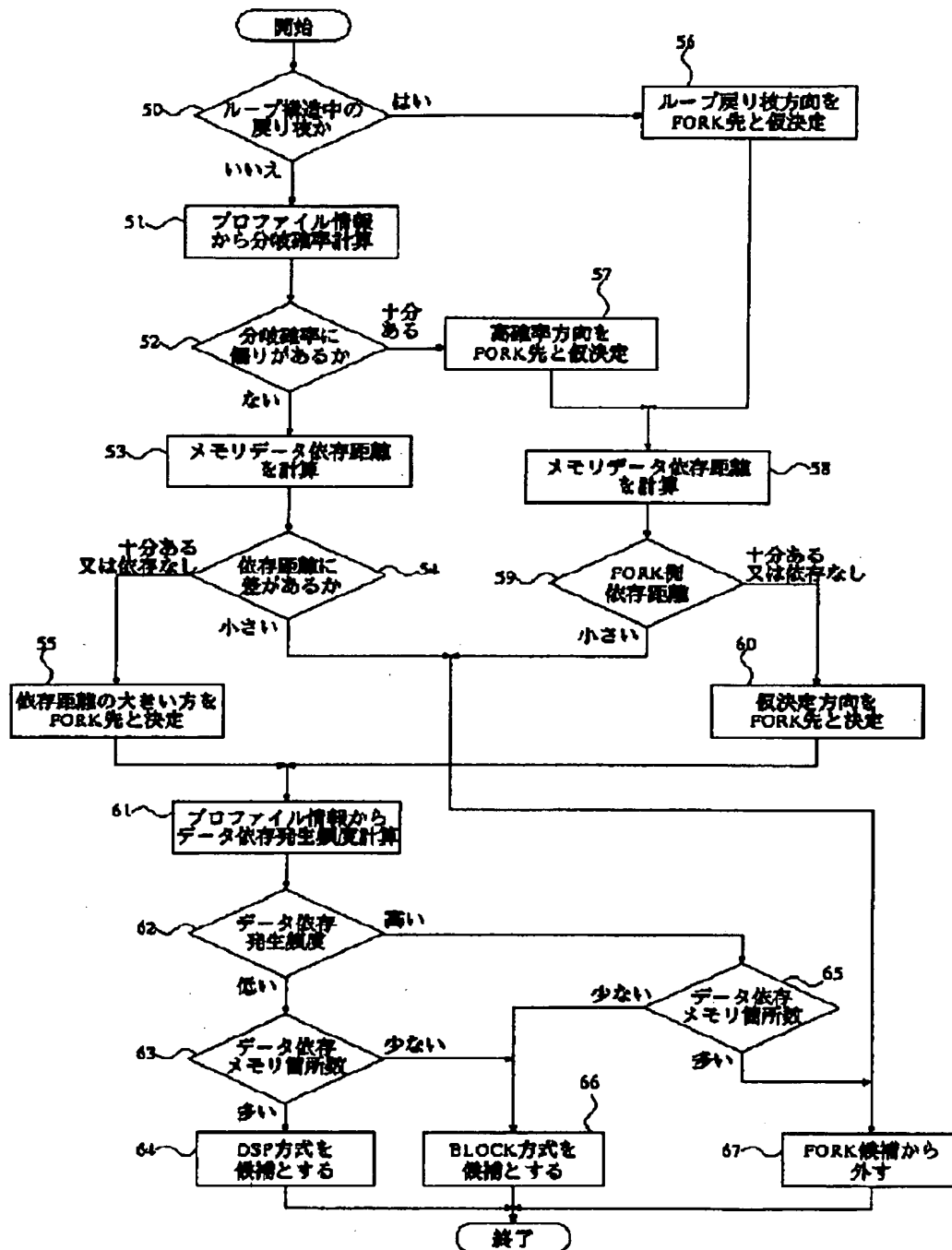
(255) x21 := AX
(256) x22 := x11
(257) x23 := 4
(258) x24 := x22 * x23
(259) x25 := x21 + x24
(260) BLOCK x25
(261) SPFORK L2
(262) x26 := 1
(263) mem(x25) := x26
(264) RELEASE x25
(265) x27 := x11
(266) x28 := 20
(267) x29 := x27 > x28
(268) PTERM
(269) THABORT
(270) goto L1
(271) L1:
(272) x20 := AX
(273) x21 := mem(6J)
(274) x22 := 4
(275) x23 := x21 * x22
(276) x24 := x20 + x23
(277) x25 := mem(x24)
(278) x26 := mem(6J)
(279) x27 := x25 + x26
(280) x12 := x27
(281) goto L3
(282) L2:
(283) x20 := x13
(284) x21 := 10
(285) x22 := x20 / x21
(286) x12 := x22
(287) x23 := AX
(288) x24 := mem(6J)
(289) x25 := 4
(290) x26 := x24 + x25
(291) x27 := x23 + x26
(292) x28 := mem(x27)
(293) x29 := x12
(294) x30 := x28 + x29
(295) x12 := x30
(296) L3:

```

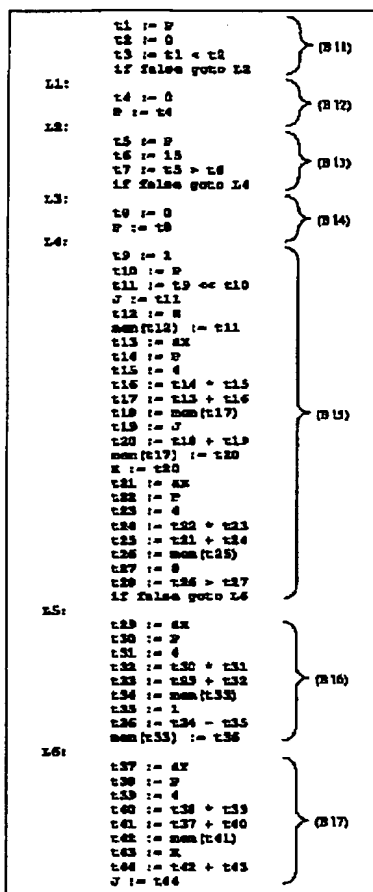
【図14】



【図13】



【図 15】



【図17】

